



Nucleus® POSIX Guide

Release 2013.08

August 2013

**© 2006-2013 Mentor Graphics Corporation
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

Chapter 1

Introduction	15
Build Configurations	15
Building with the Nucleus Dev Shell	16
Building with the Sourcery CodeBench IDE	16

Chapter 2

Nucleus POSIX Threads Library APIs	19
clock_getres	24
clock_gettime	25
clock_settime	27
kill	29
longjmp	30
mlock	32
mlockall	33
mq_close	34
mq_getattr	35
mq_notify	36
mq_open	38
mq_receive	40
mq_send	42
mq_setattr	44
mq_timedreceive	46
mq_unlink	48
munlock	49
munlockall	50
nanosleep	51
pause	53
NU_Posix_Exit_Plus_Task	54
NU_Posix_Register_Plus_Task	56
pthread_attr_destroy	58
pthread_attr_getdetachstate	59
pthread_attr_getguardsize	60
pthread_attr_getinheritsched	61
pthread_attr_getrrinterval	62
pthread_attr_getschedparam	63
pthread_attr_getschedpolicy	64
pthread_attr_getscope	65
pthread_attr_getstack	66
pthread_attr_getstackaddr	67
pthread_attr_getstacksize	68
pthread_attr_init	69

pthread_attr_setdetachstate	71
pthread_attr_setguardsize	73
pthread_attr_setinheritsched	74
pthread_attr_setrinterval	76
pthread_attr_setschedparam	77
pthread_attr_setschedpolicy	79
pthread_attr_setscope	81
pthread_attr_setstack	83
pthread_attr_setstackaddr	84
pthread_attr_setstacksize	85
pthread_cancel	86
pthread_cleanup_pop	87
pthread_cleanup_push	88
pthread_cond_broadcast	89
pthread_cond_destroy	91
pthread_cond_signal	92
pthread_cond_timedwait	94
pthread_cond_wait	96
pthread_condattr_destroy	98
pthread_condattr_getpshared	99
pthread_condattr_init	101
pthread_condattr_setpshared	102
pthread_create	104
pthread_detach	106
pthread_equal	107
pthread_exit	108
pthread_getconcurrency	109
pthread_getschedparam	110
pthread_getspecific	111
pthread_setconcurrency	112
pthread_join	113
pthread_key_create	115
pthread_key_delete	117
pthread_kill	118
pthread_mutex_destroy	121
pthread_mutex_getprioceiling	122
pthread_mutex_init	123
pthread_mutex_lock	124
pthread_mutex_setprioceiling	125
pthread_mutex_trylock	126
pthread_mutex_unlock	127
pthread_mutexattr_destroy	128
pthread_mutexattr_getprioceiling	129
pthread_mutexattr_getprotocol	130
pthread_mutexattr_getpshared	131
pthread_mutexattr_init	132
pthread_mutexattr_setprioceiling	133
pthread_mutexattr_setprotocol	134
pthread_mutexattr_setpshared	136

Table of Contents

pthread_once	137
pthread_self	138
pthread_setcancelstate	139
pthread_setcanceltype	140
pthread_setschedparam	141
pthread_setspecific	143
pthread_sigmask	144
pthread_testcancel	146
raise	147
sched_get_priority_max	148
sched_get_priority_min	149
sched_getparam	150
sched_getscheduler	151
sched_rr_get_interval	152
sched_setparam	154
sched_yield	155
sem_close	156
sem_destroy	157
sem_getvalue	158
sem_init	159
sem_open	161
sem_post	163
sem_timedwait	164
sem_trywait	166
sem_unlink	167
sem_wait	168
setjmp	169
sigaction	171
sigaddset	173
sigdelset	174
sigemptyset	175
sigfillset	176
sigismember	177
siglongjmp	179
sigpending	181
sigprocmask	182
sigqueue	184
sigsetjmp	187
sigsuspend	189
sigtimedwait	190
sigwait	192
sigwaitinfo	194
sleep	196
timer_create	199
timer_delete	201
timer_gettime	202
timer_settime	203
uname	205

Chapter 3**Nucleus POSIX Run Time Library APIs 207**

abort	214
abs	215
acos	216
Alarm	217
asctime	220
asctime_r	221
asin	223
atan	224
atan2	225
atof	226
atoi	227
atol	228
atoll	229
bcmp	230
bcopy	231
bsearch	232
bzero	234
calloc	235
ceil	236
clearerr	237
clock	238
clock_nanosleep	239
confstr	241
cos	243
cosh	244
ctime	245
ctime_r	246
difftime	247
div	248
exp	249
fabs	250
fclose	251
fdopen	253
feclearexcept	255
fegetenv	256
fegetexceptflag	257
fegetround	258
feholdexcept	259
feof	260
feraiseexcept	261
ferror	262
fesetenv	263
fesetexceptflag	264
fesetround	265
fetestexcept	266
feupdateenv	267

Table of Contents

fflush	268
ffs	269
fgetc	270
fgetpos	272
fgets	274
fileno	276
flockfile	277
floor	278
fmod	279
fopen	280
fpathconf	282
fprintf	284
fputc	287
fputs	289
fread	291
free	293
frexp	294
freopen	295
fscanf	298
fseek	300
fsetpos	302
ftell	304
ftrylockfile	306
funlockfile	307
fwrite	308
getc	310
getc_unlocked	312
getchar	314
getchar_unlocked	315
getenv	316
getpid	317
gets	318
gmtime	320
gmtime_r	321
hypot	322
imaxabs	323
imaxdiv	324
isalnum	325
isalpha	326
isascii	327
isblank	328
isctrl	329
isdigit	330
isgraph	331
islower	332
isprint	333
ispunct	334
isspace	335
isupper	336

isxdigit	337
labs	338
ldexp	339
ldiv	340
llabs	341
lldiv	342
localtime	343
localtime_r	344
log	346
log10	347
malloc	348
memchr	349
memcmp	350
memcpy	352
memmove	353
memset	354
mktime	355
modf	357
mq_timedsend	358
pathconf	362
perror	363
pow	364
printf	365
pthread_atfork	368
pthread_getcpuclockid	370
pthread_mutex_timedlock	372
pthread_mutexattr_gettype	375
pthread_mutexattr_settype	377
putc	379
putc_unlocked	381
putchar	383
putchar_unlocked	385
puts	387
qsort	389
rand	391
rand_r	392
realloc	393
remove	395
rewind	397
scanf	398
setbuf	400
setenv	402
setlocale	404
setvbuf	405
signal	407
sin	409
sinh	410
snprintf	411
sprintf	414

Table of Contents

sqrt.	416
srand	417
sscanf.	418
strcat	420
strchr	421
strcmp	422
strcoll.	423
strcpy.	425
strcspn	426
strdup.	427
strerror.	428
strerror_r	429
strftime	430
strlen	434
strncat	435
strncmp	437
strncpy.	439
strpbrk.	440
strrchr	441
strspn.	442
strstr.	443
strtod	444
strtof	445
strtoimax	446
strtok	447
strtok_r	448
strtol	449
strtoll	450
strtold	451
strtoul	452
strtoull.	453
strtoumax.	454
strxfrm.	455
swab	457
sysconf	458
tan	459
tanh	460
time	461
tmpfile.	462
tmpnam	464
toascii	466
tolower	467
toupper	468
ungetc	469
unsetenv	471
vfprintf	472
vsnprintf	475

Chapter 4

Nucleus POSIX File System Library APIs	479
access	481
chdir	483
close	484
closedir	486
creat	487
dup	489
dup2	491
fcntl	493
fdatasync	497
fseeko	498
fstat	500
fsync	502
ftello	504
ftruncate	505
getcwd	506
ioctl	508
I_ALLOCTIMEOUT	510
I_BUFFALLOC	512
I_BUFFDEALLOC	514
I_RTIMEOUT	515
I_SFLAGS	516
I_SIFFPTR	517
I_WTIMEOUT	518
link	519
lseek	520
mkdir	522
mmap	524
msync	526
munmap	528
open	529
opendir	533
read	535
readdir	537
readdir_r	539
rename	541
rewinddir	543
rmdir	544
seekdir	546
shm_open	547
shm_unlink	550
stat	551
telldir	553
unlink	554
utime	556
write	558

Chapter 5**Nucleus POSIX Networking Library APIs..... 561**

accept	563
bind	565
connect	567
endhostent	569
FD_CLR	570
FD_ISSET	571
FD_PSET	572
FD_ZERO	573
gethostbyaddr	574
gethostbyname	576
gethostent	577
gethostname	578
getpeername	579
getsockname	581
getsockopt	583
htonl	590
htons	591
if_freenameindex	592
if_indextoname	593
if_nameindex	594
if_nametoindex	595
inet_addr	596
inet_ntoa	597
inet_ntop	598
inet_pton	600
listen	602
ntohl	604
ntohs	605
recv	606
recvfrom	609
recvmsg	612
select	615
send	617
sendmsg	620
sendto	623
sethostent	626
setsockopt	627
shutdown	634
socket	636

Chapter 6**Nucleus POSIX Asynchronous I/O Library APIs..... 639**

aio_cancel	640
aio_disable	642
aio_enable	643
aio_error	645

aio_read	647
aio_return	648
aio_suspend	649
aio_write	651
lio_listio	652

Embedded Software and Hardware License Agreement

List of Tables

Table 2-1. Default Thread Attribute Values	69
Table 4-1. mman.h Header Files	526
Table 4-2. oflag Values	529
Table 5-1. getsockopt Levels	583
Table 5-2. recv Flags	606
Table 5-3. recvfrom Flags	609
Table 5-4. recvmsg Flags	612
Table 5-5. send Flags	617
Table 5-6. sendmsg Flags	620
Table 5-7. sendto Flags	623
Table 5-8. setsockopt Levels	627
Table 5-9. shutdown Flags	634
Table 5-10. Socket Types	636
Table 5-11. Protocol Types	636

Chapter 1

Introduction

Nucleus POSIX (Portable Operating System) consists of the following sets of API libraries for OS compatibility with UNIX.

- [Nucleus POSIX Threads Library APIs](#)
- [Nucleus POSIX Run Time Library APIs](#)
- [Nucleus POSIX File System Library APIs](#)
- [Nucleus POSIX Networking Library APIs](#)
- [Nucleus POSIX Asynchronous I/O Library APIs](#)

The POSIX component is part of the Nucleus Services and it is located at `\os\services\posix`. Depending on your embedded application, any of these components can be configured for use.

Note



Your source code is initially located in the `<install_root>\nucleus` directory. The source from this directory is copied into the project folder you specify.

Note



Nucleus POSIX does not support operation with MMU enabled.

Note



Nucleus POSIX does not support wide-character operations (C or C++ language).

Build Configurations

All components are by default enabled through the `.metadata` file located at the top level of each component's installation, for example for POSIX: `\os\services\posix`. When a component is enabled, it is included in the build.

You can create a user configuration file to override the default configuration and exclude a component from the build. For more information on creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

Building with the Nucleus Dev Shell

When building using the command-line interface provided by the Nucleus Dev Shell, the Nucleus OS library must be configured to include Nucleus POSIX and support the use of POSIX through changes to the compilation tools' settings. This procedure explains the configuration file modifications needed to build using the Nucleus Dev Shell.

Prerequisites

- Locate the existing custom configuration (*.config*) file that is used to build the Nucleus OS library through the Nucleus Dev Shell.

Procedure

1. Ensure that all supporting middleware is enabled in the Nucleus OS configuration.

Nucleus POSIX components rely on Nucleus Middleware for functionality. For example, the Nucleus POSIX NET component relies on Nucleus NET. If all Nucleus POSIX components are to be used, it is recommended that the Nucleus OS should be configured to have at least the core components enabled. The supporting middleware can be enabled by adding the following line to the *.config* file:

```
.include "config/core.config"
```

2. Modify the existing Nucleus OS *.config* file to add the following lines to enable Nucleus POSIX:

```
.include "config/csgnu_arm.posix.config"  
nu.os.svcs.posix.enable=true
```

3. Save changes to the Nucleus OS configuration files, clean and re-build the Nucleus OS library.

Results

A Nucleus OS library is created that can be linked with a POSIX application.

Related Topics

[Build Configurations](#)

[Building with the Sourcery CodeBench IDE](#)

Building with the Sourcery CodeBench IDE

When building using the Nucleus ReadyStart Sourcery CodeBench IDE, the Nucleus System Project used to build the Nucleus OS library, and the application project must be modified to properly support Nucleus POSIX. This includes enabling the Nucleus POSIX components and

any supporting middleware as well as updating the build settings for both the Nucleus System Project and application.

The following procedure provides guidance on how to build using the Sourcery CodeBench IDE:

Prerequisites

- A Nucleus System Project has been created in the CodeBench IDE.
- An application project has been created in the CodeBench IDE using a Nucleus application template and referencing the Nucleus System Project.

Procedure

1. Open the Nucleus System Project configuration file (*.config*) for the application using the Nucleus Configuration Editor.
 - a. Enable Nucleus POSIX and the supporting middleware (Nucleus NET and Nucleus File, for example).

Note



If enabling all Nucleus POSIX components, the core template may be used as a starting point. The core template is enabled in the Nucleus Configuration Editor of the IDE by selecting Core in the Template drop-down list.

- b. Update the toolset compiler (nu.toolset) settings:

```
CFLAGS="-isystem ${SYSTEM_HOME}/os/include/services -isystem
${SYSTEM_HOME}/os/include/services/sys -Wall -ffunction-sections -
fdata-sections"
CXXFLAGS="-isystem ${SYSTEM_HOME}/os/include/services -isystem
${SYSTEM_HOME}/os/include/services/sys -Wall -fno-enforce-eh-specs"
LDFLAGS="-nostdlib -nostartfiles -Wl,--gc-sections -Wl,--defsym -
Wl,PAGE_SIZE=${PAGE_SIZE} "
```

- c. Save the changes to the Nucleus System Project configuration file.
2. Open the application properties by right-clicking on the application in the Project Explorer view and selecting **Properties** from the context menu that appears.
 - a. Under **C/C++ Build > Settings**, on the **Tools Settings** tab, select **Codesourcery GNU C Linker** (or **Codesourcery GNU C++ Linker**) and edit the **Command Line Pattern** to be:

```
${COMMAND} ${FLAGS}
${OUTPUT_FLAG}
${OUTPUT_PREFIX}${OUTPUT} -Wl,--start-group ${INPUTS} -lgcc -Wl,--
end-group
```

- b. Under **C/C++ Build > Environment**, under Environment variables to set, edit **NUCLEUS_LIBS** to be:

`-lnucleus -lm -lstdc++`

Note

The tools C run-time library (`-lc`) is not included. This is done to prevent conflicts with the Nucleus POSIX implementations of the C run-time library functions.

- c. Under **C/C++ Build > Environment**, under Environment variables to set, edit **CFLAGS** to be:

`-isystem ${SYSTEM_HOME}/os/include/services -isystem
${SYSTEM_HOME}/os/include/services/sys -Wall -ffunction-sections -fdata-
sections`

Note

If the application project was created after the Nucleus System Project settings were updated, the application will inherit the updated build settings and no changes will be needed.

- d. Under **C/C++ Build > Environment**, under Environment variables to set, edit **CXXFLAGS** to be:

`-isystem ${SYSTEM_HOME}/os/include/services -isystem
${SYSTEM_HOME}/os/include/services/sys -Wall -fno-enforce-eh-specs`

Note

If the application project was created after the Nucleus System Project settings were updated, the application will inherit the updated build settings and no changes will be needed.

- e. Click **OK** to save the changes.
3. Close the application properties and rebuild the application project, which will also build the Nucleus System Project using the application's configuration file.

Results

The result of these instructions is an application image (`.out`) file built with Nucleus POSIX enabled.

Related Topics

[Build Configurations](#)

[Building with the Nucleus Dev Shell](#)

Chapter 2

Nucleus POSIX Threads Library APIs

This chapter provides a detailed reference of all the Nucleus POSIX Threads Library (Pthreads) APIs.

- [clock_getres](#)
- [clock_gettime](#)
- [clock_settime](#)
- [kill](#)
- [longjmp](#)
- [mlock](#)
- [mlockall](#)
- [mq_close](#)
- [mq_getattr](#)
- [mq_notify](#)
- [mq_open](#)
- [mq_receive](#)
- [mq_send](#)
- [mq_setattr](#)
- [mq_timedreceive](#)
- [mq_unlink](#)
- [munlock](#)
- [munlockall](#)
- [nanosleep](#)
- [pause](#)
- [NU_Posix_Exit_Plus_Task](#)
- [NU_Posix_Register_Plus_Task](#)
- [pthread_attr_destroy](#)

- [pthread_attr_getdetachstate](#)
- [pthread_attr_getguardsize](#)
- [pthread_attr_getinheritsched](#)
- [pthread_attr_getrrinterval](#)
- [pthread_attr_getschedparam](#)
- [pthread_attr_getschedpolicy](#)
- [pthread_attr_getscope](#)
- [pthread_attr_getstack](#)
- [pthread_attr_getstackaddr](#)
- [pthread_attr_getstacksize](#)
- [pthread_attr_init](#)
- [pthread_attr_setdetachstate](#)
- [pthread_attr_setguardsize](#)
- [pthread_attr_setinheritsched](#)
- [pthread_attr_setrrinterval](#)
- [pthread_attr_setschedparam](#)
- [pthread_attr_setschedpolicy](#)
- [pthread_attr_setscope](#)
- [pthread_attr_setstack](#)
- [pthread_attr_setstackaddr](#)
- [pthread_attr_setstacksize](#)
- [pthread_cancel](#)
- [pthread_cleanup_pop](#)
- [pthread_cleanup_push](#)
- [pthread_cond_broadcast](#)
- [pthread_cond_destroy](#)
- [pthread_cond_signal](#)
- [pthread_cond_timedwait](#)
- [pthread_cond_wait](#)

-
- [pthread_condattr_destroy](#)
 - [pthread_condattr_getpshared](#)
 - [pthread_condattr_init](#)
 - [pthread_condattr_setpshared](#)
 - [pthread_create](#)
 - [pthread_detach](#)
 - [pthread_equal](#)
 - [pthread_exit](#)
 - [pthread_getconcurrency](#)
 - [pthread_getschedparam](#)
 - [pthread_getspecific](#)
 - [pthread_setconcurrency](#)
 - [pthread_join](#)
 - [pthread_key_create](#)
 - [pthread_key_delete](#)
 - [pthread_kill](#)
 - [pthread_mutex_destroy](#)
 - [pthread_mutex_getprioceiling](#)
 - [pthread_mutex_init](#)
 - [pthread_mutex_lock](#)
 - [pthread_mutex_setprioceiling](#)
 - [pthread_mutex_trylock](#)
 - [pthread_mutex_unlock](#)
 - [pthread_mutexattr_destroy](#)
 - [pthread_mutexattr_getprioceiling](#)
 - [pthread_mutexattr_getprotocol](#)
 - [pthread_mutexattr_getpshared](#)
 - [pthread_mutexattr_init](#)
 - [pthread_mutexattr_setprioceiling](#)

- [pthread_mutexattr_setprotocol](#)
- [pthread_mutexattr_setpshared](#)
- [pthread_once](#)
- [pthread_self](#)
- [pthread_setcancelstate](#)
- [pthread_setcanceltype](#)
- [pthread_setschedparam](#)
- [pthread_setspecific](#)
- [pthread_sigmask](#)
- [pthread_testcancel](#)
- [raise](#)
- [sched_get_priority_max](#)
- [sched_get_priority_min](#)
- [sched_getparam](#)
- [sched_getscheduler](#)
- [sched_rr_get_interval](#)
- [sched_setparam](#)
- [sched_yield](#)
- [sem_close](#)
- [sem_destroy](#)
- [sem_getvalue](#)
- [sem_init](#)
- [sem_open](#)
- [sem_post](#)
- [sem_timedwait](#)
- [sem_trywait](#)
- [sem_unlink](#)
- [sem_wait](#)
- [setjmp](#)

- [sigaction](#)
- [sigaddset](#)
- [sigdelset](#)
- [sigemptyset](#)
- [sigfillset](#)
- [sigismember](#)
- [siglongjmp](#)
- [sigpending](#)
- [sigprocmask](#)
- [sigqueue](#)
- [sigsetjmp](#)
- [sigsuspend](#)
- [sigtimedwait](#)
- [sigwait](#)
- [sigwaitinfo](#)
- [sleep](#)
- [timer_create](#)
- [timer_delete](#)
- [timer_gettime](#)
- [timer_settime](#)
- [uname](#)

clock_getres

This service gets the resolution of the specified clock.

Usage

```
int clock_getres (clockid_t      clock_id,  
                 struct timespec *res)
```

Arguments

- `clock_id`
Specifies the ID of the clock whose time is to be set.
- `res`
Pointer to the `timespec` structure, to be filled with the resolution of the specified clock.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the following error code:
`EINVAL`

The `clock_id` argument does not specify a known clock.

Example

```
clockid_t clockid;  
struct timespec res;  
int ret;  
  
clockid = CLOCK_REALTIME;  
  
ret = clock_getres(clockid,&res);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [ctime](#)

[mq_timedreceive](#)

[nanosleep](#)

[sem_timedwait](#)

[time](#)

[<time.h>](#)

[timer_create](#)

[Nucleus POSIX Threads Library APIs](#)

clock_gettime

This service gets the specified clock, `clock_id`, in to the value specified by `tp`.

Usage

```
int clock_gettime (clockid_t      clock_id,  
                  struct timespec *tp)
```

Arguments

- `clock_id`
Specifies the ID of the clock whose time is to be obtained. The `clock_id` parameter can be one of the following:

`CLOCK_REALTIME`
Identifier for the system wide real-time clock.
- `tp`
Pointer to the time retrieved for the specified clock.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EINVAL`
Indicates `tp` argument is `NULL`.

`EINVAL`
The `clock_id` argument does not specify a known clock.

Example

```
clockid_t clockid;  
struct timespec tp;  
int ret;  
  
clockid = CLOCK_REALTIME;  
  
/* Get the time for the real-time clock */  
ret = clock_gettime(clock_id,&tp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[mq_timedreceive](#)

[nanosleep](#)

[sem_timedwait](#)

[time](#)

[<time.h>](#)

[timer_create](#)

[Nucleus POSIX Threads Library APIs](#)

clock_settime

This service sets the specified clock, `clock_id`, to the value specified by `tp`.

Usage

```
int clock_settime (clockid_t          clock_id,  
                  const struct timespec *tp)
```

Arguments

- `clock_id`
Specifies ID of the clock whose time is to be set. The `clock_id` parameter can be one of the following:

`CLOCK_REALTIME`
Identifier for the system wide real-time clock.
- `tp`
Pointer to the `timespec` structure, containing the time value that we are going to set for the clock.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EINVAL`
Indicates that `tp` argument specified a nanosecond value less than zero, or greater than or equal to one billion, or it is `NULL`.

`EINVAL`
The `clock_id` argument does not specify a known clock.

Example

```
clockid_t clockid;  
struct timespec tp;  
int ret;  
  
clockid = CLOCK_REALTIME;  
tp.tv_sec = 1;  
tp.tv_nsec = 0;  
  
/* Set the clock to one second */  
ret = clock_settime(clock_id,&tp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [ctime](#)

[mq_timedreceive](#)

[nanosleep](#)

[sem_timedwait](#)

[time](#)

[<time.h>](#)

[timer_create](#)

[Nucleus POSIX Threads Library APIs](#)

kill

This function sends a signal to a process or a group of processes specified by pid.

Usage

```
int kill (pid_t pid,  
          int  sig)
```

Arguments

- sig
Specifies the signal number.
- pid
Process ID.

Return Values

- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code. In the case of Nucleus, since there is no support of processes, kill() always returns POSIX_ERROR.

Example

```
int ret;  
ret = kill(0, SIGABRT);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [raise](#)

[sigaction](#) [<signal.h>](#)

[sigqueue](#) [<sys/types.h>](#)

[getpid](#) [Nucleus POSIX Threads Library APIs](#)

longjmp

This function restores the environment saved by the most recent invocation of [setjmp](#) in the same thread, with the corresponding jmp_buf [setjmp](#) argument.

Usage

```
void longjmp (jmp_buf env,  
             int    val)
```

Arguments

- env
Jump array buffer contains all the saved values.
- val
If val is 0, setjmp() returns 1.

Example

```
#include "services/stdio.h"
#include "services/stdlib.h"
#include "services/setjmp.h"

void one_time(void);
void exits(void);
jmp_buf place;

main(){
    /* First call returns 0, Longjmp() will return 4. */
    if (setjmp(place) != 0)
    {
        printf("longjmp success \n");
        exits();
    }

    one_time();
    /* This statement will never be reached - longjmp will go back above.*/
    printf("Error, something is wrong!\n");
}

void one_time(void)
{
    /* Return to setjmp, returning 4! */
    longjmp(place, 4);
    printf("Error, something is wrong!\n");
}

void exits(void)
{
    /* forever loop */
    while(1);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [setjmp](#)

[<setjmp.h>](#) [sigaction](#)

[siglongjmp](#) [sigsetjmp](#)

[Nucleus POSIX Threads Library APIs](#)

mlock

This service locks a range of process address space.

Usage

```
int mlock (const void *addr,  
          size_t      len)
```

Arguments

- **addr**
Pointer to the start of process address space.
- **len**
Length in bytes of the process address space to be locked.

Return Values

- **POSIX_SUCCESS**
Always return a success as in the case of Nucleus, process address space is always physical memory resident. There is no concept of virtual memory.

Example

```
#include "services/sys/mman.h"  
int status;  
  
/* Memory lock certain address space. Nucleus does not have the concept of  
   physical memory */  
status = mlock(addr, 4096);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <sys/mman.h>
2001

[Nucleus POSIX Threads Library APIs](#)

mlockall

This service locks the process address space.

Usage

```
int mlockall (int flags)
```

Arguments

- flags

Flags control the lock operation. This can be set to one of the following:

MCL_CURRENT

Lock currently mapped pages.

MCL_FUTURE

Lock pages that become mapped.

Return Values

- **POSIX_SUCCESS**

Indicates successful completion of the service.

- **POSIX_ERROR**

Indicates that the service fails and sets errno to the following error code:

EINVAL

Invalid value of the flags passed.

Example

```
#include "services/sys/mman.h"
int status;

/* Memory lock certain address space. Nucleus does not have the concept of
   physical memory. This API always returns success */
status = mlockall(MCL_CURRENT);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <sys/mman.h>

[Nucleus POSIX Threads Library APIs](#)

mq_close

This service removes the association between the message queue descriptor, mqdes, and its message queue.

Usage

```
int mq_close (mqd_t mqdes)
```

Arguments

- **mqdes**
Specifies the message queue descriptor of the message queue to be closed.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the following error code:
EBADF
The mqdes argument is not a valid message queue descriptor.

Example

```
mqd_t mqdes;  
int ret;  
  
/* Close the already created message queue */  
ret = mq_close(mqdes);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [mq_open](#)

[mq_unlink](#) <mqqueue.h>

[Nucleus POSIX Threads Library APIs](#)

mq_getattr

This service obtains the status information and attributes of the message queue and the open message queue description associated with the message queue descriptor.

Usage

```
int mq_getattr (mqd_t      mqdes,  
                struct mq_attr *mqstat)
```

Arguments

- **mqdes**
Specifies the message queue descriptor.
- **mqstat**
Pointer to the message queue status.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the following error code:
EBADF
The `mqdes` argument is not a valid message queue descriptor.

Example

```
mqd_t mqdes;  
struct mq_attr mqstat;  
int ret;  
  
/* Get the Message Queue attributes */  
ret = mq_getattr(mqdes, &mqstat);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[mq_open](#)

[mq_send](#)

[mq_setattr](#)

[<mqueue.h>](#)

[Nucleus POSIX Threads Library APIs](#)

mq_notify

If the argument notification is not NULL, this service registers the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, mqdes.

Usage

```
int mq_notify (mqd_t          mqdes,  
               const struct sigevent *notification)
```

Arguments

- mqdes
Specifies the message queue descriptor.
- notification
Specifies the notification to be registered.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EBADF
The mqdes argument is not a valid message queue descriptor open for reading.
 - EBUSY
A task is already registered for notification by the message queue.

Example

```
mqd_t mqdes;  
struct sigevent notification;  
int ret;  
  
notification.sigev_signo      = SIGALRM;  
notification.sigev_value.sival_int = SIGALRM;  
notification.sigev_notify     = SIGEV_SIGNAL;  
  
/* Set the notification on an empty message queue */  
ret = mq_notify(mqdes,&notification);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[mq_send](#)

[mq_open](#)
<mqueue.h>

Nucleus POSIX Threads Library APIs

mq_open

This service establishes the connection between a process and a message queue with a message queue descriptor. It creates an open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description.

Usage

```
mqd_t mq_open (char          *name,  
               int           oflag,  
               /* struct mq_attr attr */)

```

Arguments

- **name**
Specifies the name of the message queue to be opened.
- **oflag**
Specifies the flags for creating the Message Queue. The value of oflag is the bit wise-inclusive of values from the following list:
 - O_RDONLY**
Open the message queue for receiving messages.
 - O_WRONLY**
Open the queue for sending messages.
 - O_RDWR**
Open the message queue for both receiving and sending messages.
 - O_CREAT**
Create a message queue. It requires two additional arguments: mode, which is of type mode_t, and attr, which is a pointer to an mq_attr structure.
 - O_EXCL**
If O_EXCL and O_CREAT are set, mq_open fails if the message queue name exists.
 - O_NONBLOCK**
Determines whether an [mq_send](#) or [mq_receive](#) waits for resources or messages that are not currently available, or fails with errno set to [EAGAIN].
- **attr (optional)**
Specifies attributes of the message queue to be opened.

Return Values

- **mqd_t**
Return message queue descriptor upon successful completion of the service.

- **POSIX_ERROR**

Indicates that the service fails and set `errno` to the corresponding error code:

ENAMETOOLONG

The length of the name argument exceeds `{PATH_MAX}` or a pathname component is longer than `{NAME_MAX}`.

EEXIST

`O_CREAT` and `O_EXCL` are set and the named message queue already exists.

ENOENT

`O_CREAT` is not set and the named message queue does not exist.

ENFILE

Too many message queues are currently open in the system.

ENOSPC

There is insufficient space for the creation of the new message queue.

Example

```
char name[] = "queue1";
int oflag;
mode_t mode;
struct mq_attr attr;
int ret;

oflag = O_CREAT | O_RDWR;
mode = 0;

attr.mq_flags = O_NONBLOCK;
attr.mq_maxmsg = 5;
attr.mq_msgsize = 4;

/* Create a message queue */
ret = mq_open(name, oflag, mode, attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[mq_close](#)

[mq_getattr](#)

[<mqueue.h>](#)

[mq_receive](#)

[mq_send](#)

[mq_setattr](#)

[mq_timedreceive](#)

[mq_unlink](#)

[Nucleus POSIX Threads Library APIs](#)

mq_receive

This service receives the oldest of the highest priority message from the message queue specified by mqdes.

Usage

```
ssize_t mq_receive (mqd_t      mqdes,  
                   char        *msg_ptr,  
                   size_t      msg_len,  
                   unsigned int *msg_prio)
```

Arguments

- **mqdes**
Specifies the message queue descriptor from which the message is to be received.
- **msg_ptr**
To be filled with the message.
- **msg_len**
Given message length to be received.
- **msg_prio**
Pointer to the priority of the message.

Return Values

- Upon successful completion, this function returns the length of the selected message, in bytes, and the message is removed from the queue.
- **POSIX_ERROR**

If this service fails, no message is removed from the queue, and the function sets `errno` to the corresponding error code:

EBADF

The `mqdes` argument is not a valid message queue descriptor open for reading.

EMSGSIZE

The specified message buffer size, `msg_len`, is less than the message size attribute of the message queue.

EAGAIN

The `O_NONBLOCK` flag is set in the message queue description associated with `mqdes`, and the specified message queue is full.

Example

```
mqd_t mqdes;  
char msg_ptr[4];  
size_t msg_len;  
unsigned int msg_prio;  
int ret;
```



```
msg_len = 4;

/* Receive the message from an already created message queue */
ret = mq_receive(mqdes, msg_ptr, msg_len, &msg_prio);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<mqueue.h>](#)

[mq_open](#) [mq_send](#)

[time](#) [<time.h>](#)

[Nucleus POSIX Threads Library APIs](#)

mq_send

This service adds the message pointed to by the argument `msg_ptr` to the message queue specified by `mqdes`.

Usage

```
int mq_send (mqd_t      mqdes,  
             const char *msg_ptr,  
             size_t     msg_len,  
             unsigned int msg_prio)
```

Arguments

- `mqdes`
Specifies the message queue descriptor to which the message is to be sent.
- `msg_ptr`
Pointer to the message to be sent.
- `msg_len`
Length of the message to be sent.
- `msg_prio`
Priority of the message to be sent.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:
 - `EINVAL`
The value of `msg_prio` was outside the valid range.
 - `EBADF`
The `mqdes` argument is not a valid message queue descriptor open for writing.
 - `EMSGSIZE`
The specified message length, `msg_len`, exceeds the message size attribute of the message queue.
 - `EAGAIN`
The `O_NONBLOCK` flag is set in the message queue description associated with `mqdes`, and the specified message queue is full.

Example

```
mqd_t mqdes;  
char msg_ptr[] = "123";
```

```
size_t msg_len;  
unsigned int msg_prio;  
int ret;  
  
msg_len = 4;  
msg_prio = 10;  
  
/* Send the message to already created message queue */  
ret = mq_send(mqdes, msg_ptr, msg_len, msg_prio);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<mqqueue.h>](#)

[mq_open](#)

[mq_receive](#)

[mq_setattr](#)

[mq_timedreceive](#)

[time](#)

[<time.h>](#)

[Nucleus POSIX Threads Library APIs](#)

mq_setattr

This service sets attributes associated with the open message queue description referenced by the message queue descriptor specified by mqdes.

Usage

```
int mq_setattr (mqd_t          mqdes,  
               const struct mq_attr *mqstat,  
               struct mq_attr    *omqstat)
```

Arguments

- mqdes
Specifies message queue descriptor.
- mqstat
Specifies message queue status to be set.
- omqstat
Filled with the old message queue status.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:

EBADF
The mqdes argument is not a valid message queue descriptor open for writing.

EBADF
The mqdes argument is not a valid message queue descriptor.

Example

```
mqd_t mqdes;  
struct mq_attr mqstat;  
struct mq_attr omqstat;  
int ret;  
  
mqstat.mq_flags = O_CREAT | O_RDWR;  
  
/* Set the Message Queue attributes */  
ret = mq_setattr(mqdes, &mqstat, &omqstat);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <mqueue.h>
2001

[mq_open](#)

[mq_send](#)

[Nucleus POSIX Threads Library APIs](#)

mq_timedreceive

This service receives the oldest of the highest priority message from the message queue specified by mqdes.

Usage

```
ssize_t mq_timedreceive (mqd_t          mqdes,  
                        char            *msg_ptr,  
                        size_t          msg_len,  
                        unsigned        *msg_prio,  
                        const struct timespec *abs_timeout)
```

Arguments

- **mqdes**
Specifies message queue descriptor from which the message is to be received.
- **msg_ptr**
To be filled with the message.
- **msg_len**
Given message length to be received.
- **msg_prio**
Pointer to the priority of the message.
- **abs_timeout**
Pointer to the absolute timeout value.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:
 - EBADF**
The mqdes argument is not a valid message queue descriptor open for reading.
 - EMSGSIZE**
The specified message buffer size, msg_len, is less than the message size attribute of the message queue.
 - EAGAIN**
The O_NONBLOCK flag is set in the message queue description associated with mqdes, and the specified message queue is full.

EINVAL

The thread would have blocked and the `abs_timeout` parameter specified a nanoseconds value less than zero, or greater than or equal to one billion.

ETIMEDOUT

The `O_NONBLOCK` flag was not set when the message queue was opened, but no message arrived on the queue before the specified timeout expired.

Description

If `O_NONBLOCK` was not specified when the message queue was opened via the [mq_open](#) function, and no message exists on the queue to satisfy the receive, the wait for such a message will be terminated when the specified timeout expires. If `O_NONBLOCK` is set, this function is equivalent to [mq_receive](#).

Example

```
mqd_t mqdes;
char msg_ptr[4];
size_t msg_len;
unsigned int msg_prio;
struct timespec timeout;
int ret;

msg_len = 4;

/* Set the timeout value equal to 2 seconds. */
timeout.tv_sec = 2;
timeout.tv_nsec = 0;

/* Receive the message from an already created message queue */
ret = mq_timedreceive(mqdes, msg_ptr, msg_len, &msg_prio, &timeout);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<mqueue.h>](#)

[mq_open](#) [mq_send](#)

[time](#) [<time.h>](#)

[Nucleus POSIX Threads Library APIs](#)

mq_unlink

This service removes the message queue named by “name”.

Usage

```
int mq_unlink (const char *name)
```

Arguments

- **name**
Specifies the name of the message queue to be unlinked.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:
 - ENAMETOOLONG**
The length of the name argument exceeds `{PATH_MAX}` or a pathname component is longer than `{NAME_MAX}`.
 - ENOENT**
The named message queue does not exist.

Example

```
int ret;  
  
/* Unlink the already created message queue */  
ret = mq_unlink("queue1");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<mqueue.h>](#)

[mq_close](#)

[mq_open](#)

[Nucleus POSIX Threads Library APIs](#)

munlock

This service unlocks a range of process address space.

Usage

```
int munlock (const void *addr,  
            size_t      len)
```

Arguments

- **addr**
Pointer to the start of process address space.
- **len**
Length in bytes of the process address space to be unlocked.

Return Values

- **POSIX_SUCCESS**
Always return a success as in the case of Nucleus, process address space is always physical memory resident. There is no concept of virtual memory.

Example

```
#include "services/sys/mman.h"  
int status;  
  
/* Memory unlock certain address space. Nucleus does not have the concept  
   of physical memory */  
status = munlock(addr, 4096);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <sys/mman.h>
2001

[Nucleus POSIX Threads Library APIs](#)

munlockall

This service unlocks the process address space.

Usage

```
int munlockall (void)
```

Return Values

- **POSIX_SUCCESS**

Always return a success as in the case of Nucleus, process address space is always physical memory resident. There is no concept of virtual memory.

Example

```
#include "services/sys/mman.h"
int status;

/* Nucleus does not have the concept of physical memory */
status = munlockall();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <sys/mman.h>
2001

[Nucleus POSIX Threads Library APIs](#)

nanosleep

This service suspends the current thread from execution until the time interval specified by the `rqtp` argument has elapsed.

Usage

```
int nanosleep (const struct timespec *rqtp,  
               struct timespec      *rmtp)
```

Arguments

- `rqtp`
Pointer to the requested time period for the sleep.
- `rmtp`
Pointer to the remaining time period in case if the signal causes the call to return before the requested time.

Return Values

- 0
If `sleep()` returns because the requested time has elapsed.
- If `sleep()` returns due to the delivery of a signal, the return value is the "unslept" amount (the requested time minus the time actually slept) in seconds.
- `EINVAL`
The `rqtp` argument specified a nanosecond value less than zero or greater than or equal to one billion.
- `EINTR`
The `nanosleep()` function was interrupted by a signal.

Description

This function causes the current thread to be suspended from execution until either the time interval specified by the `rqtp` argument has elapsed or a signal is delivered to the calling thread. It invokes a signal-catching function or terminates the thread. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution, or because of the scheduling of other activity by the system. With the exception of being interrupted by a signal, the suspension time is not less than the time specified by `rqtp`, as measured by the system clock, `CLOCK_REALTIME`.

The use of the `nanosleep()` function has no effect on the action or blockage of any signal.

Example

```
struct timespec rqtp;  
struct timespec rmtp;  
int ret;  
  
rqtp.tv_sec = 1;
```

```
rqtp.tv_nsec = 0;

/* Sleep the calling thread for one second */
ret = nanosleep(&rqtp,&rmtp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[sleep](#)

<time.h>

[Nucleus POSIX Threads Library APIs](#)

pause

This service suspends the thread until a signal is received.

Usage

```
int pause(void)
```

Return Values

- -1

A value of -1 will be returned and `errno` set to indicate the error.



Note

Since this service suspends thread execution indefinitely unless interrupted by a signal, there is no successful completion return value.

Example

```
int status;
sigset_t set;
sigset_t save_set;

/* Empty the signal set */
sigemptyset(&set);

/* Add SIGUSR in the signal set */
sigaddset(&set, SIGUSR1);

/* sigprocmask() and pthread_sigmask() are very important API for enabling
   signals. You must call sigprocmask() or pthread_sigmask() to enable
   the signals for this thread. */

status = pthread_sigmask(SIG_UNBLOCK, &set, &save_set);

if (status != 0)
{
    printf("error\n");
}

/* suspend on pause. It will always return error. */
status = pause();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[sigsuspend](#)

<unistd.h>

[Nucleus POSIX Threads Library APIs](#)

NU_Posix_Exit_Plus_Task

This function exits the POSIX thread.

Usage

```
VOID NU_Posix_Exit_Plus_Task(VOID)
```

Description

This function handles a thread that is detached or joinable. This API must be invoked when [NU_Posix_Register_Plus_Task](#) is previously called within the same Nucleus PLUS task. This thread is set by default as joinable and performs self cleanup if [pthread_detach](#) is invoked.

Example

```
#include "services/ptc_defs.h"

void* Posix_Thread_0_Entry(void *argv);

/* PLUS task */
NU_TASK MainTask;

/* This is PLUS created task */
VOID Plus_Task_0(UNSIGNED argc, VOID *argv)
{
    pthread_t posix_thread_0;
    int status;

    /* Register PLUS Task as POSIX Thread */
    NU\_Posix\_Register\_Plus\_Task(&MainTask);

    /* Now the user is free to use any POSIX API */

    /* Spawning thread from PLUS created Task */

    status =
        pthread_create(&posix_thread_0, NULL, Posix_Thread_0_Entry, NULL);

    if (status == POSIX_SUCCESS)
    {
        /* It is now safe to perform file system operations */
        ....
    }

    /* give POSIX thread chance to run */
    sleep(1);

    /* Call an end to POSIX thread */
    NU\_Posix\_Exit\_Plus\_Task();
}

void* Posix_Thread_0_Entry(void *argv)
{
    /* POSIX thread spawn from PLUS registered task */
}
```

Note



The example assumes Nucleus FILE is included.

Related Topics

[NU_Posix_Register_Plus_Task](#)

[Nucleus POSIX Threads Library APIs](#)

NU_Posix_Register_Plus_Task

This function registers a Nucleus PLUS task as a Nucleus POSIX thread so the API can be called from a task created from Nucleus PLUS [NU_Create_Task](#).

Usage

```
int NU_Posix_Register_Plus_Task(pthread_t *thread)
```

Arguments

- thread
Thread ID of the registered thread.

Return Values

- POSIX_SUCCESS
Function call is successful.
- POSIX_ERROR
Error.

Description

This function registers PLUS Task as POSIX thread so that Nucleus POSIX API can be called from a task created from Nucleus PLUS [NU_Create_Task](#). Since the parent thread might be PLUS Task, the registered thread does not inherit scheduling parameters from the parent thread or is set to PTHREAD_EXPLICIT_SCHED. You must use Nucleus POSIX API to kill the registered thread. You can do this, for example, by calling [pthread_detach](#), [pthread_exit](#), [pthread_kill](#) or thread cancellation from its own thread context or from another thread context. This API uses the Nucleus PLUS Task Control Block tc_system_reserved_2 data member.

The current Nucleus PLUS Task is registered with the following attributes:

- Priority based on PLUS Task.
- Stack address and size based on PLUS Task.
- Scheduling policy based on PLUS Task.
- Round Robin interval based on PLUS Task.
- Inheritance as PTHREAD_INHERIT_SCHED.
- Detach state as PTHREAD_CREATE_JOINABLE, which can be changed through [pthread_detach](#).
- cancelation_state as PTHREAD_CANCEL_ENABLE, which can be changed through [pthread_setcancelstate](#).
- cancelability_type as PTHREAD_CANCEL_DEFERRED, which can be changed through [pthread_setcanceltype](#).

Example

Note



The example assumes Nucleus FILE is included.

```
#include "services/ptc_defs.h"

void* Posix_Thread_0_Entry(void *argv);

/* PLUS task */
NU_TASK MainTask;

/* This is PLUS created task */
VOID Plus_Task_0(UNSIGNED argc, VOID *argv)
{
    pthread_t posix_thread_0;
    int status;

    /* Register PLUS Task as POSIX Thread */
    NU\_Posix\_Register\_Plus\_Task(&MainTask);

    /* Now the user is free to use any POSIX API */

    /* Spawning thread from PLUS created Task */
    status = pthread_create(&posix_thread_0, NULL, Posix_Thread_0_Entry,
                           NULL);
    if (status == POSIX_SUCCESS)
    {
        /* It is now safe to perform file system operations */
        ....
    }
    /* give POSIX thread chance to run */
    sleep(1);

    /* Call an end to POSIX thread */
    NU\_Posix\_Exit\_Plus\_Task();
}

void* Posix_Thread_0_Entry(void *argv)
{
    /* POSIX thread spawn from PLUS registered task */
}
```

Related Topics

[NU_Posix_Exit_Plus_Task](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_destroy

This service removes the storage allocated during initialization. The attribute object becomes invalid.

Usage

```
int pthread_attr_destroy (pthread_attr_t *attr)
```

Arguments

- attr
Pointer to the attribute object to be destroyed.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of attr is not valid.

Example

```
pthread_attr_t attr;  
int ret;  
  
/* Destroy the attributes, which are already initialized */  
ret = pthread_attr_destroy(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_getdetachstate](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_getstacksize](#)

[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getdetachstate

This service retrieves the thread create state, which can either be detached or joined.

Usage

```
int pthread_attr_getdetachstate (const pthread_attr_t *attr,  
                                int *detachstate)
```

Arguments

- attr
Pointer to a given attribute object.
- detachstate
Pointer to the detach state.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of detachstate is NULL or attr is invalid.

Example

```
pthread_attr_t attr;  
int detachstate;  
int ret;  
  
/* Get DetachState of a thread attribute */  
ret = pthread_attr_getdetachstate(&attr,&detachstate);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_getstacksize](#)


[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getguardsize

This function retrieves the guard area size value from the specified thread create attribute object.

 **Note** The check_stack_enabled configuration option must be set to true for the functionality of stack checking on the guarded area of the stack to work.

Usage

```
int pthread_attr_getguardsize (const pthread_attr_t *attr,  
                             size_t                *guardsize)
```

Arguments

- attr
Pointer to a thread create attribute object.
- guardsize
Pointer to the memory that is updated with the object's guard area size value if the operation is successful.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates whether the value of the guardsize parameter is NULL, or the thread create attribute object is invalid.

Example

```
pthread_attr_t attr;  
size_t guardsize;  
int ret;  
/* Get guard area size of a thread attribute */  
ret = pthread_attr_getguardsize(&attr,&guardsize);
```

Related Topics

[pthread_attr_destroy](#)

[pthread_attr_getstacksize](#)

[pthread_attr_getstack](#)

[<pthread.h>](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_setguardsize](#)

[pthread_create](#)

Base Definitions volume of IEEE std 1003.1 - 2001

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getinheritsched

This service retrieves the thread scheduling policy set in the attribute object.

Usage

```
int pthread_attr_getinheritsched (const pthread_attr_t *attr,  
                                int *inheritsched)
```

Arguments

- **attr**
Pointer to a given attribute object.
- **inheritsched**
Specifies whether or not the scheduling parameters are to be inherited from the parent thread.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Indicates the value of policy is NULL or attr is invalid.

Example

```
pthread_attr_t attr;  
int inheritsched;  
int ret;  
  
/* Get the inheritsched parameter of a thread attribute */  
ret = pthread_attr_getinheritsched(&attr,&inheritsched);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getschedparam](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getrrinterval

This service retrieves the round robin interval that was set in the attribute object.

Usage

```
int pthread_attr_getrrinterval (const pthread_attr_t *attr,  
                               struct timespec      *interval)
```

Arguments

- attr
Pointer to a given attribute object.
- interval
To be filled with the round robin interval.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Either the attr or interval is NULL or the attr is invalid.

Example

```
pthread_attr_t attr;  
struct timespec interval;  
int ret;  
  
/* Get the Round Robin interval */  
ret = pthread_attr_getrrinterval(&attr, &interval);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getschedparam](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getschedparam

This service retrieves the thread scheduling parameters set in the attribute object.

Usage

```
int pthread_attr_getschedparam (const pthread_attr_t *attr,  
                                struct sched_param  *param)
```

Arguments

- attr
Pointer to a given attribute object.
- param
Pointer to the thread scheduling parameter attributes.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of param is NULL or attr is invalid.

Example

```
pthread_attr_t attr;  
struct sched_param param;  
int ret;  
  
/* Get the scheduling parameter of a thread attribute */  
ret = pthread_attr_getschedparam(&attr,&param);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getinheritsched](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getschedpolicy

This service retrieves the thread scheduling policy set in the attribute object.

Usage

```
int pthread_attr_getschedpolicy (const pthread_attr_t *attr,  
                                int *policy)
```

Arguments

- attr
Pointer to a given attribute object.
- policy
Pointer to the scheduling policy.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of policy is NULL or attr is invalid.

Example

```
pthread_attr_t attr;  
int policy;  
int ret;  
  
/* Get the scheduling policy of a thread attribute */  
ret = pthread_attr_getschedpolicy(&attr,&policy);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getinheritsched](#)

[pthread_attr_getschedparam](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getscope

This service retrieves the thread scope, which indicates whether the thread is scheduled at the system level or at the user level.

Usage

```
int pthread_attr_getscope (const pthread_attr_t *attr,  
                           int *contentionscope)
```

Arguments

- attr
Pointer to a given attribute object.
- contentionscope
Pointer to the contentionscope.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of scope is NULL or attr is invalid.

Example

```
pthread_attr_t attr;  
int scope;  
int ret;  
  
/* Get the contentionscope of a thread attribute */  
ret = pthread_attr_getscope(&attr,&scope);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getinheritsched](#)

[pthread_attr_getschedparam](#)

[pthread_attr_setschedpolicy](#)

[pthread_create](#)

<pthread.h>

<sched.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getstack

This function retrieves the stack attribute values (address and size) from the specified thread create attribute object.

Usage

```
int pthread_attr_getstack (const pthread_attr_t *attr,  
                           void **stackaddr,  
                           size_t *stacksize)
```

Arguments

- **attr**
Pointer to a thread create attribute object.
- **stackaddr**
Pointer to memory that is updated with the object's stack address value if the operation is successful.
- **stacksize**
Pointer to memory that is updated with the object's stack address value if the operation is successful.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Indicates that the thread create attribute object is invalid, or that the stackaddr or stacksize parameters are invalid.

Example

```
pthread_attr_t attr;  
void * stackaddr;  
size_t stacksize;  
int ret;  
/* Get stack characteristics of a thread attribute */  
ret = pthread_attr_getstack(&attr,&stackaddr,&stacksize);
```

Related Topics

[pthread_attr_setstack](#)

[pthread_attr_destroy](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_getstacksize](#)

[pthread_attr_getguardsize](#)

[pthread_create](#)

[<pthread.h>](#)

Base Definitions volume of IEEE Std 1003.1 - 2001

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getstackaddr

This service retrieves the stack address that was set in the attribute object.

Usage

```
int pthread_attr_getstackaddr (const pthread_attr_t *attr,  
                               void **stackaddr)
```

Arguments

- attr
Pointer to a given attribute object.
- stackaddr
Pointer to the stack address.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of stackaddr or attr is incorrect.

Example

```
pthread_attr_t attr;  
void *base;  
int ret;  
  
/* Get the stack address */  
ret = pthread_attr_getstackaddr(&attr,&base);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <limits.h>

[pthread_attr_destroy](#)

[pthread_attr_getdetachstate](#)

[pthread_attr_getstacksize](#)

[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_getstacksize

This service retrieves the stack size set in the attribute object.

Usage

```
int pthread_attr_getstacksize (pthread_attr_t *attr,  
                               size_t         *stacksize)
```

Arguments

- attr
Pointer to a given attribute object.
- stacksize
Pointer to the stack size.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of attr is invalid or either attr or stacksize is NULL.

Example

```
pthread_attr_t attr;  
size_t size;  
int ret;  
  
/* Get the stack size */  
ret = pthread_attr_getstacksize(&attr, &size);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <limits.h>
2001

[pthread_attr_destroy](#)

[pthread_attr_getdetachstate](#)

[pthread_attr_getstackaddr](#)

[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_init

This service initializes a thread attribute object attr with the default value for all of the individual attributes used.

Usage

```
int pthread_attr_init (pthread_attr_t *attr)
```

Arguments

- attr
Pointer to the thread attribute object to be filled with the default thread attributes.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- ENOMEM
Indicates that there is not enough memory to initialize the thread attributes object.

Description

The individual attributes and their default settings are described in [Table 2-1](#).

Table 2-1. Default Thread Attribute Values

Attribute	Default Value
priority	Default value is 127. Medium priority of 127 is assigned to a new thread.
kernel_stack_addr	Default value is NULL. A new thread has a system-allocated stack address.
kernel_stack_size	Default value is PTHREAD_STACK_MIN (2KBytes). A new thread has a system-defined stack size.
inheritsched	Default value is PTHREAD_INHERIT_SCHED. A new thread inherits parent thread scheduling parameters.
contention_scope	Default value is PTHREAD_SCOPE_SYSTEM. A new thread has the kernel thread scheduling scope. Nucleus does not support PTHREAD_SCOPE_PROCESS.
schedpolicy	Default value is SCHED_FIFO. Preemptive, priority based scheduling.
rrinterval	Default value is 100000000 nsecs. Round robin interval equal to 10 ticks of Nucleus PLUS for time slicing in SCHED_RR policy.
detach_state	Default value is PTHREAD_CREATE_JOINABLE. Exit status and thread ID are preserved after the thread terminates.

Example

```
pthread_attr_t attr;  
int ret;  
  
/* Create the main thread for the initializations purposes */  
ret = pthread_attr_init(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_getdetachstate](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_getstacksize](#)

[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setdetachstate

This service sets the detach_state member in the thread attributes.

Usage

```
int pthread_attr_setdetachstate (pthread_attr_t *attr,  
                                int detachstate)
```

Arguments

- attr
Pointer to the attribute object.
- detachstate
Specifies one of the following detachstates to be set in the attribute object.

PTHREAD_CREATE_DETACHED

The thread ID and other resources can be reused as soon as the thread terminates.

PTHREAD_CREATE_JOINABLE

It is assumed that the user will wait for PTHREAD_CREATE_JOINABLE, otherwise there will be memory leaks for the resources occupied by that thread.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that the value of detachstate or attr is not valid.

Example

```
pthread_attr_t attr;  
pthread_t tid;  
extern void start_routine(void*);  
void *arg;  
int ret;  
  
/* Initialize the thread with default attributes */  
arg = NULL;  
  
ret = pthread_attr_init(&attr);  
ret = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);  
  
ret = pthread_create(&tid, &attr, start_routine, arg);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getstackaddr](#)

[pthread_create](#)

[Nucleus POSIX Threads Library APIs](#)

[pthread_attr_getstacksize](#)

[<pthread.h>](#)

pthread_attr_setguardsize

This function sets the guard area size value in the specified thread create attribute object.

Note



The `check_stack_enabled` configuration option must be set to true for the functionality of stack checking on the guarded area of the stack to work.

Usage

```
int pthread_attr_setguardsize (pthread_attr_t *attr,  
                             size_t          guardsize)
```

Arguments

- `attr`
Pointer to a thread create attribute object.
- `guardsize`
The guard area size value to be set in the object.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `EINVAL`
Indicates the thread create attribute object is invalid.

Example

```
pthread_attr_t attr;  
int ret;  
  
/* Set guard area size of a thread attribute */  
ret = pthread_attr_setguardsize(&attr,1024);
```

Related Topics

[pthread_attr_destroy](#)

[pthread_attr_setstack](#)

[pthread_attr_getguardsize](#)

[<pthread.h>](#)

[pthread_attr_getstackaddr](#)

[pthread_attr_getstack](#)

[pthread_create](#)

Base Definitions volume of IEEE Std 1003.1 - 2001

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setinheritsched

This service sets the inheritsched attribute in the attr object.

Usage

```
int pthread_attr_setinheritsched (pthread_attr_t *attr,  
                                int inheritsched)
```

Arguments

- attr
Pointer to a given attribute object.
- inheritsched
Specifies whether or not the scheduling parameters are to be inherited from the parent thread. This can be one of the following settings:
 - PTHREAD_INHERIT_SCHED
Scheduling policies defined in the creating thread are to be used, and any scheduling attributes defined in the [pthread_create](#) call are to be ignored.
 - PTHREAD_EXPLICIT_SCHED
Attributes from the [pthread_create](#) call are to be used.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that an attempt was made to set attr to a value that is not valid.

Example

```
pthread_attr_t attr;  
pthread_t tid;  
extern void start_routine(void*);  
void *arg;  
int ret;  
  
/* Initialize the thread with default attributes */  
arg = NULL;  
  
ret = pthread_attr_init(&attr);  
ret = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);  
  
ret = pthread_create(&tid, &attr, start_routine, arg);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getschedparam](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setrrinterval

This service sets the round robin interval attribute of the attr argument.

Usage

```
int pthread_attr_setrrinterval (pthread_attr_t      *attr,  
                               const struct timespec *interval)
```

Arguments

- attr
Pointer to a given attribute object.
- interval
Specifies the round robin interval to be set in the attribute object.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that either value of the attr or interval is invalid.

Example

```
pthread_attr_t attr;  
struct timespec interval;  
int ret;  
  
interval.tv_sec = 0;  
interval.tv_nsec = 1000000000;  
  
/* Set the new Round Robin interval in the attribute object */  
ret = pthread_attr_setrrinterval(&attr, &interval);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getschedparam](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setschedparam

This service sets thread scheduling parameters in the attr object that are defined in the param structure.

Usage

```
int pthread_attr_setschedparam (pthread_attr_t      *attr,  
                               const struct sched_param *param)
```

Arguments

- attr
Pointer to a given attribute object.
- param
Pointer to the thread scheduling parameters to set.
For SCHED_FIFO and SCHED_RR scheduling policies, the only parameter is priority.
Newly created threads run with this priority.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates the value of param or attr is invalid or if the priority given in param structure is greater than POSIX_PRIORITY_MAX.

Example

```
pthread_attr_t attr;  
int newprio;  
struct sched_param param;  
int ret;  
  
newprio = 30;  
  
/* Set the priority in the sched_param structure */  
param.sched_priority = newprio;  
  
/* Set the new scheduling parameter */  
ret = pthread_attr_setschedparam(&attr,&param);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getinheritsched](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getscope](#)

[pthread_create](#)

<pthread.h>

<sched.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setschedpolicy

This service sets the scheduling policy attribute in the attr object.

Usage

```
int pthread_attr_setschedpolicy (pthread_attr_t *attr,  
                                int             policy)
```

Arguments

- attr
Pointer to a given attribute object.
- policy
Specifies the scheduling policy to be set in the attribute object. The policy parameter can be one of the following:

SCHED_FIFO

First in first out scheduling policy.

SCHED_RR

Round robin scheduling policy.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that an attempt is made to set attr to a value that is not valid.
- ENOTSUP
SCHED_OTHER is given in the policy argument.

Example

```
pthread_attr_t attr;  
pthread_t tid;  
extern void start_routine(void*);  
void *arg;  
int ret;  
  
/* Initialize the thread with default attributes */  
arg = NULL;  
  
ret = pthread_attr_init(&attr);  
ret = pthread_attr_setschedpolicy(&attr, SCHED_RR);  
ret = pthread_create(&tid, &attr, start_routine, arg);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_attr_getinheritsched](#)

[pthread_attr_getscope](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

[pthread_attr_destroy](#)

[pthread_attr_getschedparam](#)

[pthread_create](#)

[<sched.h>](#)

pthread_attr_setscope

This service sets the contention scope attribute in the attr object.

Usage

```
int pthread_attr_setscope (pthread_attr_t *attr,  
                           int             contention_scope)
```

Arguments

- attr
Pointer to a given attribute object.
- contention_scope
Specifies the scheduling contention_scope to be set in the attribute object. The contention_scope parameter can be one of the following:

PTHREAD_SCOPE_SYSTEM

Threads are scheduled at the system level.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that an attempt is made to set attr to a value that is not valid.
- ENOTSUP
PTHREAD_SCOPE_PROCESS is given in the contention_scope argument.

Example

```
pthread_attr_t attr;  
pthread_t tid;  
extern void start_routine(void*);  
void *arg;  
int ret;  
  
/* Initialize the thread with default attributes */  
arg = NULL;  
  
ret = pthread_attr_init(&attr);  
ret = pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);  
ret = pthread_create(&tid, &attr, start_routine, arg);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_attr_getinheritsched](#)

[pthread_attr_getschedpolicy](#)

[pthread_attr_getschedparam](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

[pthread_create](#)

[<sched.h>](#)

pthread_attr_setstack

This function sets the stack attribute values (address and size) from the specified thread create attribute object.

Usage

```
int pthread_attr_setstack (pthread_attr_t *attr,  
                           void          *stackaddr,  
                           size_t        stacksize)
```

Arguments

- **attr**
Pointer to a thread create attribute object.
- **stackaddr**
Stack address value to be set in the thread create attribute object.
- **stacksize**
Stack size value to be set in the thread create attribute object.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Indicates that the thread create attribute object is invalid, or that the stackaddr or stacksize parameters are invalid.

Example

```
pthread_attr_t attr;  
int ret;  
  
/* Set stack characteristics of a thread attribute */  
ret = pthread_attr_setstack(&attr, 0x80000000, 1024);
```

Related Topics

[pthread_attr_setstack](#)[pthread_attr_destroy](#)[pthread_attr_getstackaddr](#)[pthread_attr_getstacksize](#)[pthread_attr_getstack](#)[pthread_attr_getguardsize](#)[pthread_create](#)[<pthread.h>](#)

Base Definitions volume of IEEE Std 1003.1 - 2001 [Nucleus POSIX Threads Library APIs](#)

pthread_attr_setstackaddr

This service defines the base of the thread's stack. If this is set to non-null (NULL is the default) the system initializes the stack at that address.

Usage

```
int pthread_attr_setstackaddr (pthread_attr_t *attr,  
                               void            *stackaddr)
```

Arguments

- attr
Pointer to a given attribute object.
- stackaddr
Pointer to the stack address to be set in the attribute object.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Indicates that the value of stackaddr or attr is incorrect.

Example

```
pthread_attr_t attr;  
void *base;  
int ret;  
  
base = (void*)malloc(PTHREAD_STACK_Min + 0x4000);  
  
/* Set the new stack address in the attribute object */  
ret = pthread_attr_setstackaddr(&attr, base);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <limits.h>

[pthread_attr_destroy](#)

[pthread_attr_getdetachstate](#)

[pthread_attr_getstacksize](#)

[pthread_create](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_attr_setstacksize

This service defines the size of the stack (in bytes) that the system allocates. The size should not be less than the system-defined minimum stack size.

Usage

```
int pthread_attr_setstacksize (pthread_attr_t *attr,  
                               size_t          stacksize)
```

Arguments

- **attr**
Pointer to a given attribute object.
- **stacksize**
Specifies the stack size to be set in the attribute object.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Indicates the size given is less than the value of **PTHREAD_STACK_MIN**, or exceeds a system-imposed limit, or **attr** is not valid.

Example

```
pthread_attr_t attr;  
size_t size;  
int ret;  
  
size = PTHREAD_STACK_MIN + 0x4000;  
  
/* Set the new stack size in the attribute object*/  
ret = pthread_attr_setstacksize(&attr, size);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <limits.h>
2001

[pthread_attr_destroy](#)

[pthread_attr_getdetachstate](#)

[pthread_attr_getstackaddr](#)

[pthread_create](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_cancel

This service cancels the target thread.

Usage

```
int pthread_cancel (pthread_t thread)
```

Arguments

- **thread**
ID of the target thread to be canceled.

Return Values

- **ESRCH**
No thread could be found corresponding to that specified by the given thread ID.
- **POSIX_SUCCESS**
Indicates successful completion of the service.

Description

How the cancellation request is treated depends on the state of the target thread. Two functions, [pthread_setcancelstate](#) and [pthread_setcanceltype](#), determine that state.

Example

```
pthread_t tid;  
int ret;  
  
/* Cancel the target thread indicated by tid */  
ret = pthread_cancel(tid);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_timedwait](#)

[pthread_exit](#)

[pthread_join](#)

[pthread_setcancelstate](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_cleanup_pop

This service pulls the cleanup handler off the cleanup stack.

Usage

```
void pthread_cleanup_pop (int execute)
```

Arguments

- `execute`
 - `nonzero`
Removes the handler from the stack and executes it.
 - `0`
Removes the handler from the stack without executing it.

Description

This service is effectively called with a nonzero argument if a thread either explicitly or implicitly calls [pthread_exit](#) or if the thread accepts a cancel request.

Example

```
int execute;

/* Pull the cancellation routine from the cancellation stack and
   execute it */

execute = 1;
pthread_cleanup_pop(execute);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cancel](#)

[pthread_setcancelstate](#) [<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_cleanup_push

This service pushes a cleanup handler onto a cleanup stack (LIFO).

Usage

```
void pthread_cleanup_push (void (*routine) (void*),  
                           void *arg)
```

Arguments

- **routine**
Specifies the function pointer to the cleanup routine.
- **arg**
Argument passed to the cleanup routine.

Example

```
extern void routine(void*);  
void arg;  
  
arg = NULL;  
  
/* Push the cancellation routine on the cancellation stack */  
pthread_cleanup_push(routine, arg);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cancel](#)

[pthread_setcancelstate](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_cond_broadcast

This service unblocks all the threads that are blocked on the specified condition variable cond.

Usage

```
int pthread_cond_broadcast (pthread_cond_t *cond)
```

Arguments

- cond
Pointer to the condition variable.

Return Values

- EINVAL
The value cond does not refer to an initialized condition variable.
- POSIX_SUCCESS
Indicates successful completion of the service.

Description

If more than one thread is blocked on a condition variable then scheduling policy will determine the order in which the threads will be unblocked. When each thread is unblocked as a result of this call the thread will own the mutex with which it called [pthread_cond_wait](#) or [pthread_cond_timedwait](#).

A thread may call this function whether or not it currently owns the mutex that threads calling [pthread_cond_wait](#) or [pthread_cond_timedwait](#) have associated with the condition variable during their waiting state. However if the predictable result is required, then the mutex will be locked by the thread calling [pthread_cond_signal](#).

Example

```
pthread_cond_t cond;  
int status;  
  
/* Create the condition variable object */  
...  
  
/* Wait on the condition variable */  
...  
  
/* Broadcast the condition variable */  
status = pthread_cond_broadcast(&cond);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_cond_timedwait](#)

[pthread_cond_destroy](#)
<pthread.h>

Nucleus POSIX Threads Library APIs

pthread_cond_destroy

This service is used to destroy the given condition variable specified by cond. The object becomes, in effect, uninitialized.

Usage

```
int pthread_cond_destroy (pthread_cond_t *cond)
```

Arguments

- cond
Pointer to the condition variable to be destroyed.

Return Values

- EINVAL
The value specified by cond is invalid.
- EBUSY
An attempt to destroy the object referenced by cond while it is referenced by another thread.
- POSIX_SUCCESS
Indicates successful completion of the service.

Example

```
pthread_cond_t cond;  
int status;  
  
/* Create the condition variable object */  
...  
  
/* Destroy the condition variable object */  
status = pthread_cond_destroy(&cond);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_broadcast](#)

[pthread_cond_signal](#)

[pthread_cond_timedwait](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_cond_signal

This service unblocks at least one of the threads that are blocked on the specified condition variable `cond` (if any threads are blocked on `cond`).

Usage

```
int pthread_cond_signal (pthread_cond_t *cond)
```

Arguments

- `cond`
Pointer to the condition variable.

Return Values

- `EINVAL`
The value `cond` does not refer to an initialized condition variable.
- `POSIX_SUCCESS`
Indicates successful completion of the service.

Description

If more than one thread is blocked on a condition variable then scheduling policy will determine the order in which the threads will be unblocked. When each thread is unblocked as a result of this call the thread will own the mutex with which it called [pthread_cond_wait](#) or [pthread_cond_timedwait](#).

A thread may call this function whether or not it currently owns the mutex that threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waiting state. However if the predictable result is required, then the mutex will be locked by the thread calling [pthread_cond_signal](#).

Example

```
pthread_cond_t cond;
int status;

/* Create the condition variable object */
...

/* Wait on the condition variable */
...

/* Signal the condition variable */
status = pthread_cond_signal(&cond);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_destroy](#)

[pthread_cond_timedwait](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_cond_timedwait

This service blocks on a condition variable.

Usage

```
int pthread_cond_timedwait (pthread_cond_t      *cond,  
                           pthread_mutex_t      *mutex,  
                           const struct timespec *abstime)
```

Arguments

- **cond**
Pointer to the condition variable.
- **mutex**
Pointer to the mutex.
- **abstime**
Pointer to the specified absolute time.

Return Values

- **EINVAL**
The value specified by cond or mutex is invalid.
- **EPERM**
The current thread did not own mutex at the time of the call.
- **ETIMEDOUT**
The time specified by abstime has passed.
- **POSIX_SUCCESS**
Indicates successful completion of the service.

Description

This should be called with mutex locked by the calling thread or undefined behavior results. Error is returned if the absolute time specified by abstime passes before the condition cond is signaled or broadcasted, or if the absolute time specified by abstime has already been passed at the time of the call.

This function atomically releases mutex and causes the calling thread to block on the condition variable cond.

Example

```
pthread_cond_t cond;  
pthread_mutex_t mutex;  
struct timespec abstime;  
int status;
```

```
/* Create the condition variable object */
...

/* Create and lock the mutex */
...

/* Set the time out of 1 sec */
abstime.tv_sec = 1;
abstime.tv_nsec = 0;

/* Block on the condition variable */
status = pthread_cond_timedwait(&cond,&mutex,
                                (const struct timespec*)&abstime);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_broadcast](#)

[pthread_cond_signal](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_cond_wait

This service blocks on a condition variable.

Usage

```
int pthread_cond_wait (pthread_cond_t *cond,  
                      pthread_mutex_t *mutex)
```

Arguments

- **cond**
Pointer to the condition variable.
- **mutex**
Pointer to the mutex.

Return Values

- **EINVAL**
The value specified by cond or mutex is invalid.
- **EPERM**
The current thread did not own Mutex at the time of the call.
- **POSIX_SUCCESS**
Indicates successful completion of the service.

Description

This should be called with mutex locked by the calling thread or undefined behavior results.

This function automatically releases mutex and causes the calling thread to block on the condition variable cond.

Example

```
pthread_cond_t cond;  
pthread_mutex_t mutex;  
int status;  
  
/* Create the condition variable object */  
...  
  
/* Create and lock the mutex */  
...  
  
/* Block on the condition variable */  
status = pthread_cond_wait(&cond,&mutex);
```


Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_broadcast](#)

[pthread_cond_signal](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_condattr_destroy

This service destroys a condition variable attribute object and uninitializes the object.

Usage

```
int pthread_condattr_destroy (pthread_condattr_t *attr)
```

Arguments

- **attr**
Pointer to the condition variable attribute object to be destroyed.

Return Values

- **EINVAL**
The value specified by **attr** is invalid.
- **POSIX_SUCCESS**
Indicates successful completion of the service.

Example

```
pthread_condattr_t attr;  
int status;  
  
/* Destroy the condition variable attributes object */  
status = pthread_condattr_destroy(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_condattr_getpshared](#)

[pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutex_destroy](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_condattr_getpshared

This service obtains the value of the process-shared attribute from the attribute object referenced by attr.

Usage

```
int pthread_condattr_getpshared (const pthread_condattr_t *attr,  
                                int *pshared)
```

Arguments

- attr
Pointer to a condition attribute object.
- pshared
Pointer to the pshared value.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
Either attr or pshared is NULL or the attribute is not already initialized.

Example

```
pthread_condattr_t attr;  
pthread_cond_t cond;  
int pshared;  
int status;  
  
status = pthread_condattr_init(&attr);  
if (status != POSIX_SUCCESS)  
    printf("error\n");  
  
status = pthread_cond_init(&cond, &attr);  
if (status != POSIX_SUCCESS)  
    printf("error\n");  
  
/* Test only PTHREAD_PROCESS_PRIVATE since it is the only one supported */  
status = pthread_condattr_setpshared(&attr, PTHREAD_PROCESS_PRIVATE);  
if (status != POSIX_SUCCESS)  
    printf("error\n");  
  
status = pthread_condattr_getpshared(&attr, &pshared);  
if (status != POSIX_SUCCESS && pshared != PTHREAD_PROCESS_PRIVATE)  
    printf("error\n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_condattr_destroy](#)

[pthread_cond_destroy](#)

[pthread_mutex_destroy](#)

[Nucleus POSIX Threads Library APIs](#)

[pthread_create](#)

[<pthread.h>](#)

pthread_condattr_init

This service initializes a condition variable attribute object attr with the default value for all of the attributes defined by the implementation.

Usage

```
int pthread_condattr_init (pthread_condattr_t *attr)
```

Arguments

- attr
Pointer to the condition variable attribute object.

Return Values

- ENOMEM
Insufficient memory exists to initialize the condition variable attributes object.
- POSIX_SUCCESS
Indicates successful completion of the service.

Example

```
pthread_condattr_t attr;  
int status;  
  
/* Initialize the condition variable attributes object */  
status = pthread_condattr_init(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_attr_destroy](#)

[pthread_condattr_getpshared](#)

[pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutex_destroy](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_condattr_setpshared

This service is used to set the process-shared attribute in an initialized attributes object referenced by attr.

Usage

```
int pthread_condattr_setpshared (pthread_condattr_t *attr,  
                                int pshared)
```

Arguments

- attr
Pointer to a condition attribute object.
- pshared
Specifies the pshared value to be set in the condition attributes.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- ENOTSUP
The pshared value not supported.
- EINVAL
The new value specified for attribute is outside the range of legal values for that attribute.

Example

```
pthread_condattr_t attr;  
pthread_cond_t cond;  
int pshared;  
int status;  
  
status = pthread_condattr_init(&attr);  
if (status != POSIX_SUCCESS)  
    printf("error\n");  
  
status = pthread_cond_init(&cond, &attr);  
if (status != POSIX_SUCCESS)  
    printf("error\n");  
  
/* test only PTHREAD_PROCESS_PRIVATE since it is the only one supported */  
status = pthread_condattr_setpshared(&attr, PTHREAD_PROCESS_PRIVATE);  
if (status != POSIX_SUCCESS)  
    printf("error\n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_condattr_destroy](#)

[pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutex_destroy](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_create

Allowed from: Application Initialize, HISR, POSIX Thread and signal handler

This service creates a new thread, with attributes specified by attr, within a parent thread.

Usage

```
int pthread_create (pthread_t          *thread,  
                   const pthread_attr_t *attr,  
                   void                *(*start_routine) (void *),  
                   void                *arg)
```

Arguments

- **thread**
Pointer to the thread ID to be filled with the ID of the created thread. All subsequent requests made to this thread require this thread ID.
- **attr**
Pointer to the supplied thread attributes.
- **start_routine**
Specifies the starting routine of the thread.
- **arg**
A pointer that may be used to pass information to the task.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Indicates the task attributes specified by the attr parameter are invalid.
- **ENOTSUP**
If SCHED_OTHER is given in the attr -> schedpolicy. Only the SCHED_OTHER scheduling policy is supported.
- **EAGAIN**
The system lacked the necessary resources to create another thread or the system-imposed limit on the total number of threads {PTHREAD_THREADS_MAX} would be exceeded.

Description

The signal state of the new thread is initialized as follows:

- The signal mask is inherited from the creating thread.
- The set of signals pending for the new thread is empty.

When service is successful, the ID of the thread created is stored in the location referred to as thread.

Creating a thread using a NULL attribute argument has the same effect as using a default attribute; both create a default thread. When attr is initialized, it acquires the default behavior.

Example

```
pthread_t thread;
pthread_attr_t attr;
void *start_routine(void *arg);
void *arg;
int ret;
arg = NULL;

/* Initialize the attributes with the default attributes */
pthread_attr_init(&attr);

/* Create a thread whose starting point is the function "start_routine"
   and that has the default thread attributes and pass NULL as an argument
   to "start_routine." */

ret = pthread_create(&thread, &attr, start_routine, arg);

/* At this point ret indicates if the service was successful. */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_exit](#)

[pthread_join](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_detach

This service indicates that the implementation and storage for a thread can be reclaimed when thread terminates. If the thread is not terminated, this service will not cause it to terminate.

Usage

```
int pthread_detach (pthread_t thread)
```

Arguments

- **thread**
Specifies the ID of the target thread to be detached.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **ESRCH**
No thread could be found corresponding to that specified by the given thread ID.
- **EINVAL**
The implementation has detected that the value specified by thread does not refer to a joinable thread.

Example

```
pthread_t thread;  
int ret;  
  
ret = pthread_detach(thread);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_join](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_equal

This service compares the thread identification numbers of two threads.

Usage

```
int pthread_equal (pthread_t tid1,  
                  pthread_t tid2)
```

Arguments

- tid1
First thread ID to be compared.
- tid2
Second thread ID to be compared.

Return Values

- 0
The thread IDs are not identical.
- 1
The thread IDs are identical.

Example

```
pthread_t tid1;  
pthread_t tid2;  
int result;  
  
tid1 = pthread_self();  
tid2 = 4;  
  
/* Compare the two thread ID's */  
result = pthread_equal(tid1,tid2);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_create](#)

[pthread_self](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_exit

This service terminates the calling thread and make the value value_ptr available to any successful join with the terminating thread.

Usage

```
void pthread_exit (void *value_ptr)
```

Arguments

- value_ptr
Pointer to the exit status of the calling thread.

Description

Any cancellation cleanup handlers that remain on the stack are popped off and executed.

After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, the appropriate destructor functions are called in an unspecified order.

Example

```
int status;  
  
/* Set the exit status */  
status = 3;  
  
/* Terminate the calling thread itself */  
pthread_exit( (void*)&status );
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_create](#)

[pthread_join](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_getconcurrency

This function retrieves the current level of thread concurrency for an application.

Note



This API has no effect in the Nucleus POSIX implementation, but it conforms in syntax and behavior, as required by the POSIX standard.

Usage

```
int pthread_getconcurrency (void)
```

Return Values

- Returns the value set by a previous call to the [pthread_setconcurrency](#) function. If the `pthread_setconcurrency` function was not previously called, `pthread_getconcurrency` returns zero to indicate that the implementation is maintaining the concurrency level.

Example

```
int concurrency_level;

/* Get the current concurrency level of the application. */
Concurrency_level = pthread_getconcurrency();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_getschedparam

This service gets the scheduling parameters of the existing thread. In the case of scheduling policies the SCHED_FIFO and SCHED_RR the only scheduling parameter is priority.

Usage

```
int pthread_getschedparam (pthread_t      thread,  
                           int            *policy,  
                           struct sched_param *param)
```

Arguments

- **thread**
ID of the thread whose parameters are to be gotten.
- **policy**
Pointer to the scheduling policy of the target thread.
- **param**
Pointer to the scheduling parameters of the target thread.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The value of either policy or param is NULL.
- **ESRCH**
The value specified by thread does not refer to an existing thread.

Example

```
pthread_t tid;  
struct sched_param param;  
int policy;  
int ret;  
  
/* Get the scheduling parameters of the target thread */  
tid = pthread_self();  
  
ret = pthread_getschedparam(tid, &policy, &param);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <pthread.h>

[sched_getparam](#) <sched.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_getspecific

This service gets the calling thread's binding for key and stores it in the location pointed to by value.

Usage

```
void* pthread_getspecific (pthread_key_t key)
```

Arguments

- key
Given key for accessing the thread specific data.

Return Values

- (VOID*)
Filled with the calling thread's binding for the key.
- (VOID*) EINVAL
The given key is invalid.

Example

```
pthread_key_t key;  
void *value;  
  
/* The key is created previously */  
value = pthread_getspecific(key);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_key_create](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_setconcurrency

This function sets the current level of thread concurrency for an application.

Note



This API has no effect in the Nucleus POSIX implementation, but it conforms in syntax and behavior, as required by the POSIX standard.

Usage

```
int pthread_setconcurrency (int new_level)
```

Arguments

- `new_level`
The new concurrency level to be set.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `EINVAL`
Indicates the value specified by the `new_level` parameter is negative (invalid).
- `EAGAIN`
Indicates that the specified new level would cause a system resource to be exceeded.

Example

```
int ret;  
  
/* Set the concurrency level of the application. */  
ret = pthread_setconcurrency(1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [Nucleus POSIX Threads Library APIs](#)

pthread_join

This service suspends the execution of the calling thread until the target thread terminates, unless the target thread has already terminated.

Usage

```
int pthread_join (pthread_t thread,  
                 void **value_ptr)
```

Arguments

- **thread**
Specifies the ID of the target thread to join.
- **value_ptr**
To be filled with the exit status of the target thread.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **ESRCH**
No thread could be found corresponding to that specified by the given thread ID.
- **EINVAL**
The implementation has detected that the value specified by thread does not refer to a joinable thread.
- **EDEDALK**
A deadlock was detected or the value of thread specifies the calling thread.

Description

The specified thread must be in the current process and must not be detached.

When **value_ptr** is not **NULL**, it points to a location that is set to the exit status of the terminated thread when service returns successfully.

Multiple threads cannot wait for the same thread to terminate. If the threads attempt to wait, one thread returns successfully and the others fail with an error of **ESRCH**.

After this service returns, any stack storage associated with the thread can be reclaimed by the application.

Example

```
pthread_t thread;  
int ret;  
void *ret_status;
```

```
/* Waiting to join thread "tid" with status */  
  
ret = pthread_join(thread,&ret_status);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_create](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_key_create

This service allocates a key used to identify thread-specific data in a process. The key is global to all threads in the process, and all threads initially have the value NULL associated with the key when it is created.

Usage

```
int pthread_key_create (pthread_key_t  *key,  
                       void            (*destructor) (void*) )
```

Arguments

- **key**
Pointer to the key ID.
- **destructor**
Pointer to the destructor function. This destructor calls when the thread terminates.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EAGAIN**
The system-imposed limit on the total number of keys per process {PTHREAD_KEYS_MAX} has been exceeded.

Description

This service is called once for each key before the key is used. There is no implicit synchronization.

Once a key has been created, each thread can bind a value to the key. The values are specific to the thread and are maintained for each thread independently. The per-thread binding is deallocated when a thread terminates if the key was created with a destructor function.

Example

```
pthread_key_t key;  
int ret;  
  
/* Creates the key without destructor */  
  
ret = pthread_key_create(&key, NULL);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_getspecific](#)

[pthread_key_delete](#)

[<pthread.h>](#)

Nucleus POSIX Threads Library APIs

pthread_key_delete

This service destroys an existing thread-specific data key.

Usage

```
int pthread_key_delete (pthread_key_t key)
```

Arguments

- **key**
Specifies the key to be destroyed.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The given key is invalid.

Description

Once a key has been deleted, any reference to it with the [pthread_setspecific](#) or [pthread_getspecific](#) call results in the **EINVAL** error.

You should free any thread-specific resources before calling the delete function. This function does not invoke any of the destructors.

Example

```
pthread_key_t key;  
int ret;  
  
/* Deletes the previously created key */  
ret = pthread_key_delete(key);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_key_create](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_kill

This function request that a signal be delivered to the specified thread. If sig is zero, error checking is performed but no signal is actually sent.

Usage

```
int pthread_kill (pthread_t thread,  
                  int      sig)
```

Arguments

- thread
ID of the thread to which the signal is to be sent.
- sig
Signal number of the signal to be sent.

Return Values

- POSIX_SUCCESS
Function call is successful.
- ESRCH
No thread could be found corresponding to that specified by the given thread ID.
- EINVAL
The value of the sig argument is an invalid or unsupported signal number.
- EAGAIN
No resources are available to queue the signal.

Example

```
/* current implementation the user must teardown File system or  
   Networking connection manually. */  
  
int      sock0;  
pthread_t Thread_Server;  
  
/* Server thread accepting connection */  
void* thread_server_entry(void *argv)  
{  
    struct sockaddr_in serv_addr;  
    socklen_t len;  
    int sock1;  
    struct sigaction act;  
    sigset_t set;  
    sigset_t save_set;  
  
    /* SIGUSR1 is by default pre-set to kill thread, we don't want  
       this, we want to re-wire SIGUSR1 to teardown the current TCP  
       connection first then kill this thread. */
```

```
act.sa_sigaction = signal_handler;
act.sa_flags      = 0;
act.sa_mask       = 0;
sigaction(SIGUSR1, &act, NULL);

/* Let SIGUSR1 through, Now the user have SIGUSR1, SIGKILL,
   SIGSTOP and SIGCONT signals available to get through to
   interrupt this thread. */
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
pthread_sigmask(SIG_UNBLOCK, &set, &save_set);

/* Assuming the user has perform all the necessary steps of creating
   socket, bind and listen for TCP connection. */
...

/* An example where the server thread is hang forever and need to be
   destroyed! The accept connection is interrupted and signal handler is
   invoked. */
sock1 = accept(sock0, (struct sockaddr*)&serv_addr, &len);

...
}

void* thread_client_entry(void *argv)
{
    /* Signal thread_server_entry. SIGUSR1 has been configured to invoke
       signal_handler(). Important: Keep in mind that multiple signals
       calls to the Thread_Server will be queued in the order the signal
       is received. */
    pthread_kill(Thread_Server, SIGUSR1);

    /* Give below signal catching function chance to run. */
    sleep(1);
}

/* Signal handler for tearing down TCP connection */
void signal_handler(int signo)
{
    /* shutdown() can be be used to tear down
       socket recv() or send() connection */

    /* close the socket, it is also possible to close()
       read(), write(), recv(), send(), recvfrom(), etc. */

    if (close(sock0) == -1)
    {
        printf("error\n");
    }

    /* Perform any other necessary cleanup */
    ...

    /* Destroy the server thread all together after done all the necessary
       cleanup. If pthread_exit() is not invoked, the user will continue
       execution of thread_server_entry(). */
}
```

```
    pthread_exit (NULL) ;  
}
```

Caution



With the current implementation, it is very important that you close the socket or file descriptor when signaling a suspended networking or file system operation.

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [kill](#)

[pthread_self](#)

[raise](#)

[<signal.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_destroy

This service destroys the mutex object referenced by mutex.

Usage

```
int pthread_mutex_destroy (pthread_mutex_t *mutex)
```

Arguments

- **mutex**
Pointer to Pointer to the mutex object to be destroyed.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The value of mutex given is NULL or is invalid.
- **EBUSY**
An attempt is made to destroy the object referenced by mutex while it is locked or referenced.

Example

```
pthread_mutex_t mutex;  
int ret;  
  
/* Destroy an already initialized semaphore */  
ret = pthread_mutex_destroy(&mutex);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutexattr_getprioceiling](#)

[pthread_mutexattr_getpshared](#)

[pthread_mutex_lock](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_getprioceiling

This service returns the current priority ceiling of the mutex.

Usage

```
int pthread_mutex_getprioceiling (const pthread_mutex_t *mutex,  
                                int *prioceiling)
```

Arguments

- **mutex**
Pointer to the mutex object whose priority ceiling value is to be received.
- **prioceiling**
Pointer to the prioceiling value.

Return Values

- **POSIX_SUCCESS**
Indicate successful completion of the service.
- **EINVAL**
The value of prioceiling is NULL or the value specified by mutex does not refer to a currently existing mutex.

Example

```
pthread_mutex_t mutex;  
int prioceiling;  
int ret;  
  
/* Get the priority ceiling of an already initialized semaphore */  
ret = pthread_mutex_getprioceiling(&mutex,&prioceiling);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutex_destroy](#)

[pthread_mutex_lock](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_init

This service initializes the mutex referenced by mutex with attributes specified by attr.

Usage

```
int pthread_mutex_init (pthread_mutex_t      *mutex,  
                       const pthread_mutexattr_t *attr)
```

Arguments

- mutex
Pointer to the mutex object to be initialized.
- attr
Pointer to the mutex attribute object.

Return Values

- POSIX_SUCCESS
Indicate successful completion of the service.
- EINVAL
The value of mutex given is NULL or the attr is invalid.

Example

```
pthread_mutexattr_t attr;  
pthread_mutex_t mutex;  
int ret;  
  
/* Initialize the mutex with the default attributes */  
ret = pthread_mutexattr_init(&attr);  
ret = pthread_mutex_init(&mutex,&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutexattr_getprioceiling](#)

[pthread_mutexattr_getshared](#)

[pthread_mutex_lock](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_lock

This service locks the mutex object referenced by mutex. If the mutex is already locked, the calling thread will block until the mutex becomes available.

Usage

```
int pthread_mutex_lock (pthread_mutex_t *mutex)
```

Arguments

- **mutex**
Pointer to the mutex object to be locked.

Return Values

- **POSIX_SUCCESS**
Indicate successful completion of the service.
- **EINVAL**
The value specified by mutex does not refer to an initialized mutex object or is invalid.
- **EDEADLK**
The current thread already owns the mutex.
- **EINVAL**
The mutex was created with the protocol attribute having the value **PTHREAD_PRIO_PROTECT** and the calling thread's priority is higher than the mutex's current priority ceiling.

Example

```
pthread_mutex_t mutex;  
int ret;  
  
/* Lock an already initialized semaphore */  
ret = pthread_mutex_lock(&mutex);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_mutex_destroy](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_setprioceiling

This service sets the prioceiling value of the given mutex.

Usage

```
int pthread_mutex_setprioceiling (pthread_mutex_t *mutex,  
                                int prioceiling,  
                                int *old_ceiling)
```

Arguments

- **mutex**
Pointer to the mutex object whose priority ceiling value is to be received.
- **prioceiling**
Contains the priority ceiling value to be set.
- **old_ceiling**
Pointer to the old value of prioceiling.

Return Values

- **POSIX_SUCCESS**
Indicate successful completion of the service.
- **EINVAL**
The priority requested by prioceiling is out of range.
- **EINVAL**
The value specified by mutex does not refer to a currently existing mutex.

Example

```
pthread_mutex_t mutex;  
int prioceiling;  
int old_ceiling;  
int ret;  
  
prioceiling = 220;  
  
/* Set the priority ceiling of an already initialized semaphore */  
ret = pthread_mutex_setprioceiling(&mutex, prioceiling, &old_ceiling);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutex_destroy](#)

[pthread_mutex_lock](#) [<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_trylock

This service locks the mutex object referenced by the mutex. If the mutex is already locked, the call will return immediately.

Usage

```
int pthread_mutex_trylock (pthread_mutex_t *mutex)
```

Arguments

- **mutex**
Pointer to the mutex object to try to be locked.

Return Values

- **POSIX_SUCCESS**
Indicate successful completion of the service.
- **EINVAL**
The value specified by mutex does not refer to an initialized mutex object or is invalid.
- **EDEADLK**
The current thread already owns the mutex.
- **EINVAL**
The mutex was created with the protocol attribute having the value **PTHREAD_PRIO_PROTECT** and the calling thread's priority is higher than the mutex's current priority ceiling.
- **EBUSY**
The mutex could not be acquired because it was already locked.

Example

```
pthread_mutex_t mutex;  
int ret;  
  
/* Try to lock an already initialized semaphore */  
ret = pthread_mutex_trylock(&mutex);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutex_destroy](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutex_unlock

This service releases the mutex object referenced by mutex.

Usage

```
int pthread_mutex_unlock (pthread_mutex_t *mutex)
```

Arguments

- **mutex**
Pointer to mutex object to be unlocked.

Return Values

- **POSIX_SUCCESS**
Indicate successful completion of the service.
- **EINVAL**
The value specified by mutex does not refer to an initialized mutex object or is invalid.
- **EPERM**
The current thread does not own the mutex.
- **EAGAIN**
The mutex is not locked and an attempt is made to unlock it.

Example

```
pthread_mutex_t mutex;  
int ret;  
  
/* Unlock an already initialized semaphore */  
ret = pthread_mutex_unlock(&mutex);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_mutex_destroy](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_destroy

This service destroys the mutex attribute object. The object becomes, in effect, uninitialized.

Usage

```
int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)
```

Arguments

- attr
Pointer to a mutex attribute object.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- EINVAL
The value of attr given is invalid or NULL.

Example

```
pthread_mutexattr_t attr;  
int ret;  
  
/* Destroy an already initialized mutex attribute */  
ret = pthread_mutexattr_destroy(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutexattr_destroy](#)

[pthread_mutex_destroy](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_getprioceiling

This service gets the priority-ceiling attribute of a mutex attribute object pointed to by attr.

Usage

```
int pthread_mutexattr_getprioceiling (
                                const pthread_mutexattr_t *attr,
                                int *prioceiling)
```

Arguments

- attr
Pointer to the mutex attribute object.
- prioceiling
Pointer to the prioceiling value.

Return Values

- POSIX_SUCCESS
Indicate successful completion of the service.
- EINVAL
The value specified by attr or prioceiling is invalid.

Example

```
pthread_mutexattr_t attr;
int prioceiling;
int ret;

/* Get the Mutex prioceiling value */
ret = pthread_mutexattr_getprioceiling(&attr,&prioceiling);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutex_destroy](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_getprotocol

This service gets the protocol attribute of a mutex attributes object pointed to by attr.

Usage

```
int pthread_mutexattr_getprotocol (pthread_mutexattr_t *attr,  
                                  int *protocol)
```

Arguments

- attr
Pointer to the mutex attribute object.
- protocol
Pointer to the protocol value.

Return Values

- POSIX_SUCCESS
Indicate successful completion of the service.
- EINVAL
The value of attr or protocol is NULL or attr is invalid.

Example

```
pthread_mutexattr_t attr;  
int protocol;  
int ret;  
  
/* Get the Mutex protocol */  
ret = pthread_mutexattr_getprotocol(&attr,&protocol);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_create](#)

[<pthread.h>](#)

[pthread_cond_destroy](#)

[pthread_mutex_destroy](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_getpshared

This service obtains the value of the process-shared attribute from the attributes object referenced by attr.

Usage

```
int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr,  
                                int *pshared)
```

Arguments

- attr
Pointer to a mutex attribute object.
- pshared
Pointer to the pshared value.

Return Values

- 0
Service completed successfully.
- EINVAL
Either attr or pshared is NULL or attribute is not already initialized.

Example

```
pthread_mutexattr_t attr;  
int pshared;  
  
if (pthread_mutexattr_init(&attr) != POSIX_SUCCESS)  
{  
    printf("error\n");  
}  
  
status = pthread_mutexattr_getpshared(&attr,&pshared);  
if (status != POSIX_SUCCESS && pshared != PTHREAD_PROCESS_PRIVATE)  
{  
    printf("error\n");  
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutexattr_destroy](#)

[pthread_mutex_destroy](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_init

This service initializes a mutex attributes object `attr` with the default value for all of the attributes defined by the implementation.

Usage

```
int pthread_mutexattr_init (pthread_mutexattr_t *attr)
```

Arguments

- `attr`
Pointer to a mutex attribute object.

Default Attribute Values

- `protocol`
The simple protocol in which the suspension is in priority order. By default, this is `PTHREAD_PRIO_NONE`.
- `prioceiling`
The priority ceiling in case the protocol is set to `PTHREAD_PRIO_PROTECT`. By default, this is `POSIX_PRIORITY_MAX`.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `EINVAL`
The value of `attr` is `NULL`.

Example

```
pthread_mutexattr_t attr;  
int ret;  
  
/* Initialize the mutex attribute to the default mutex attribute */  
ret = pthread_mutexattr_init(&attr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_cond_destroy](#)

[pthread_create](#)

[pthread_mutexattr_destroy](#)

[pthread_mutex_destroy](#)

[<pthread.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_setprioceiling

This service sets the priority-ceiling attribute of a mutex attribute object pointed to by attr.

Usage

```
int pthread_mutexattr_setprioceiling (pthread_mutexattr_t *attr,  
                                     int prioceiling)
```

Arguments

- attr
Pointer to a mutex attribute object.
- prioceiling
Specifies the prioceiling value to be set in the mutex attributes.

Return Values

- POSIX_SUCCESS
Indicate successful completion of the service.
- EINVAL
The value specified by attr or prioceiling is invalid.

Example

```
pthread_mutexattr_t attr;  
int prioceiling;  
int ret;  
  
prioceiling = 200;  
  
/* Set the Mutex prioceiling in the mutex attribute object */  
ret = pthread_mutexattr_setprioceiling(&attr,prioceiling);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_create](#)

[<pthread.h>](#)

[pthread_cond_destroy](#)

[pthread_mutex_destroy](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_setprotocol

This service sets the protocol attribute of a mutex attributes object pointed to by attr.

Usage

```
int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr,  
                                  int protocol)
```

Arguments

- attr
Pointer to the mutex attribute object.
- protocol
Specifies the protocol value to be set in the mutex attributes. This can be set to one of the following values:
 - PTHREAD_PRIO_NONE
Specifies thread priority and scheduling will not be affected by its mutex ownership.
 - PTHREAD_PRIO_INHERIT
Specifies the task will execute at the higher of its priority or the priority of the highest priority thread waiting on any of the mutexes owned by this thread and initialized with this protocol.
 - PTHREAD_PRIO_PROTECT
Specifies the task will execute at the higher of its priority or the highest of the priority ceilings of all the mutexes owned by this thread and initialized with this attribute.

Return Values

- POSIX_SUCCESS
Indicate successful completion of the service.
- EINVAL
The value specified by attr or protocol is invalid.

Example

```
pthread_mutexattr_t attr;  
int protocol;  
int ret;  
  
protocol = PTHREAD_PRIO_INHERIT;  
  
/* Set the Mutex protocol in the mutex attribute object */  
ret = pthread_mutexattr_setprotocol(&attr, protocol);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_create](#)

<pthread.h>

[pthread_cond_destroy](#)

[pthread_mutex_destroy](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_mutexattr_setpshared

This service sets the process-shared attribute in an initialized attributes object referenced by attr.

Usage

```
int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr,  
                                int pshared)
```

Arguments

- attr
Pointer to the mutex attribute object.
- pshared
Specifies the pshared value to be set in the mutex attributes.

Return Values

- 0
Service completed successfully.
- EINVAL
The new value specified for the attribute is outside the range of legal values for that attribute.

Example

```
#include "services/pthread.h"

pthread_mutexattr_t attr;

/* Initialize the mutex attribute */
if (pthread_mutexattr_init(&attr) != POSIX_SUCCESS)
    printf("error\n");

/* Set the mutex attribute process shared to private */
if (pthread_mutexattr_setpshared(&attr, PTHREAD_PROCESS_PRIVATE) !=
    POSIX_SUCCESS)
    printf("error\n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cond_destroy](#)
[pthread_create](#) [pthread_mutexattr_destroy](#)
[pthread_mutex_destroy](#) [<pthread.h>](#)
[Nucleus POSIX Threads Library APIs](#)

pthread_once

This service calls an initialization routine the first time when this service is called. Subsequent calls to this service will have no effect.

Usage

```
int pthread_once (pthread_once_t *once_control,  
                 void (*init_routine) (void) )
```

Arguments

- **once_control**
Pointer to a flag to determine whether the associated initialization routine has been called.
- **init_routine**
Pointer to the `init_routine` function to be called.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
Either `once_control` or `init_routine` is `NULL`.

Example

```
pthread_once_t once_control;  
int ret;  
  
extern void init_routine(void);  
  
once_control = PTHREAD_ONCE_INIT  
  
ret = pthread_once(&once_control, init_routine);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_self

This service gets the ID of the calling thread.

Usage

```
pthread_t pthread_self (void)
```

Return Values

- ID of the calling thread.

Example

```
pthread_t tid;  
  
tid = pthread_self();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_create](#)

[pthread_equal](#) <pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_setcancelstate

This service enables or disables thread cancellation. When a thread is created, thread cancellation is enabled by default.

Usage

```
int pthread_setcancelstate (int state,  
                           int *oldstate)
```

Arguments

- **state**
The calling thread is set to this state.
- **oldstate**
Pointer to the old state.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The value indicated by the priority or param is not valid.
- **ENOTSUP**
An attempt was made to set the thread attribute to an unsupported value. This will happen if **SCHED_OTHER** is given in the policy.
- **ESRCH**
The value specified by thread does not refer to an existing thread.

Example

```
int state;  
int oldstate;  
int ret;  
  
/* Set cancel state of the calling thread to PTHREAD_CANCEL_DISABLE */  
state = PTHREAD_CANCEL_DISABLE;  
  
ret = pthread_setcancelstate(state,&oldstate);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_cancel](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_setcanceltype

This service sets the cancellation type to either deferred or asynchronous mode.

Usage

```
int pthread_setcanceltype (int type,  
                           int *oldtype)
```

Arguments

- **type**
Contains cancel type to be set for the calling thread.
- **oldtype**
Pointer to the old cancel type of the calling thread.

Return Values

- **EINVAL**
Either the oldtype is NULL or the type is not PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.
- **POSIX_SUCCESS**
Indicates successful completion of the service.

Description

When a thread is created, the cancellation type is set to deferred mode by default. In deferred mode, the thread can be cancelled only at cancellation points. In asynchronous mode, a thread can be cancelled at any point during its execution. Using asynchronous mode is discouraged.

Example

```
int type;  
int oldtype;  
int ret;  
  
/* Set cancellation type of the calling thread to  
   PTHREAD_CANCEL_ASYNCHRONOUS */  
type = PTHREAD_CANCEL_ASYNCHRONOUS;  
  
ret = pthread_setcanceltype(type,&oldtype);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001
<pthread.h>

[pthread_cancel](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_setschedparam

This service sets the scheduling policy and parameters of individual threads.

Usage

```
int pthread_setschedparam (pthread_t      thread,  
                           int            policy,  
                           const struct sched_param *param)
```

Arguments

- **thread**
ID of the thread whose parameters are to be set.
- **policy**
Scheduling policy for the thread. The supported policies are as follows:
 SCHED_FIFO
 SCHED_RR
- **param**
Scheduling parameters for the thread.

In the case of the **SCHED_FIFO** scheduling policy, the only scheduling parameter is priority. In the case of the **SCHED_RR** scheduling policy, the two scheduling parameters such as priority and **rrinterval** are set.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The value indicated by the priority or param is not valid.
- **ENOTSUP**
An attempt was made to set the thread attribute to an unsupported value. This will happen if **SCHED_OTHER** is given in the policy.
- **ESRCH**
The value specified by thread does not refer to an existing thread.

Example

```
pthread_t tid;  
struct sched_param param;  
int policy;  
int ret;  
  
/* Initialize the variables */  
param.sched_priority = 200;  
policy = SCHED_RR;
```

```
tid = pthread_self();  
ret = pthread_setschedparam(tid, policy, &param);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[<pthread.h>](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

pthread_setspecific

This service sets the thread-specific binding to the specified thread-specific data key.

Usage

```
int pthread_setspecific (pthread_key_t key,  
                        const void *value)
```

Arguments

- **key**
Given key for accessing the thread specific data.
- **value**
Pointer to the value to be set.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **EINVAL**
The given key is invalid.

Example

```
pthread_key_t key;  
int value;  
int ret;  
  
/* The key is created previously */  
value = 1234;  
  
ret = pthread_setspecific(key, (void*)&value);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_key_create](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

pthread_sigmask

This function examines or changes (or both) the signal mask of the calling thread.

Note



This API is very important for enabling signals within the current thread.

Usage

```
int pthread_sigmask (int          how,  
                    const sigset_t *set,  
                    sigset_t      *oset)
```

Arguments

- **how**

Indicates the way in which the set is changed. This can be set to one of the following values:

SIG_BLOCK

Union of the current set and the signal set pointed out by set.

SIG_SETMASK

Signal set pointed out by set.

SIG_UNBLOCK

Intersection of the current set and the complement of the signal set pointed out by set.

- **set**

Pointer to the given input set.

- **oset**

Pointer to the output old set.

Return Values

- **POSIX_SUCCESS**

Indicates successful completion of the service.

- **EINVAL**

The value of the how argument is not equal to one of the defined values.

Example

```
sigset_t oset;  
sigset_t set;  
int status;  
  
/* Empty the signal set */  
sigempty(&set);  
  
/* Add SIGRTMIN in the set */  
sigaddset(&set, SIGRTMIN);
```



```
/* Set the calling thread's signal mask */  
status = pthread_sigmask(SIG_SETMASK, &set, &oset);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sigaction](#)

[sigaddset](#) [sigdelset](#)

[sigemptyset](#) [sigfillset](#)

[sigismember](#) [sigpending](#)

[sigqueue](#) [sigsuspend](#)

[<signal.h>](#) [Nucleus POSIX Threads Library APIs](#)

pthread_testcancel

This service establishes a cancellation point for a calling thread.

Usage

```
void pthread_testcancel (void)
```

Description

The service is effective when thread cancellation is enabled and in deferred mode. Calling this service while cancellation is disabled has no effect.

Be careful to insert this service only in sequences where it is safe to cancel a thread. In addition to establishing cancellation points through this service, the pthreads standard specifies several cancellation points.

Example

```
/* Set the cancellation point for the calling thread */  
  
pthread_testcancel();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pthread_cancel](#)

<pthread.h>

[Nucleus POSIX Threads Library APIs](#)

raise

This function sends the signal `sig` to the executing thread. If a signal handler is called this function will not return until after the signal handler does.

Usage

```
int raise (int sig)
```

Arguments

- `sig`
Specifies the signal number.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:
 - `ESRCH`
No thread found corresponding to that specified by the given thread ID.
 - `EINVAL`
Invalid signal number to be sent.
 - `EAGAIN`
No resources are available to queue the signal.

Example

```
int status;

/* Send the abort signal to the executing thread. */
status = raise(SIGABRT);

/* Be careful, SIGABRT is predefined to kill this thread. Therefore, this
signal will kill current thread. You can re-wire this signal to take
another action using sigaction. */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[sigaction](#)

[<sys/types.h>](#)

[<signal.h>](#)

[Nucleus POSIX Threads Library APIs](#)

sched_get_priority_max

This service gets the maximum priority for the scheduling policy specified by policy.

Usage

```
int sched_get_priority_max (int policy)
```

Arguments

- **policy**
Specifies the scheduling policy for which the maximum limit on priority is to be set. The policy parameter can be set to one of the following:

SCHED_FIFO

First In First Out scheduling policy.

SCHED_RR

Round robin scheduling policy.

Return Values

- The maximum limit on priority for the given policy if the function call is successful.
- **POSIX_ERROR**

Indicates that the service fails and sets `errno` to the corresponding error code:

ENOTSUP

If **SCHED_OTHER** is given in the policy argument.

EINVAL

The value of the policy parameter does not represent a defined scheduling policy.

Example

```
int policy;  
int maxpriority;  
  
policy = SCHED_FIFO;  
  
/* Get the minimum limit on priority for the given scheduling policy */  
maxpriority = sched_get_priority_max(policy);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[sched_getscheduler](#)

[sched_rr_get_interval](#)

[sched_setparam](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

sched_get_priority_min

This service gets the minimum priority for the scheduling policy specified by policy.

Usage

```
int sched_get_priority_min (int policy)
```

Arguments

- policy

Specifies the scheduling policy for which the minimum limit on priority is set. The policy parameter can be set to one of the following:

SCHED_FIFO

First In First Out scheduling policy.

SCHED_RR

Round robin scheduling policy.

Return Values

- The minimum limit on priority for the given policy if the function call is successful.
- POSIX_ERROR

Indicates that the service fails and sets errno to the corresponding error code:

ENOTSUP

If SCHED_OTHER is given in the policy argument.

EINVAL

The value of the policy parameter does not represent a defined scheduling policy.

Example

```
int policy;  
int minpriority;  
  
policy = SCHED_FIFO;  
  
/* Get the minimum limit on priority for the given scheduling policy */  
minpriority = sched_get_priority_min(policy);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[sched_getscheduler](#)

[sched_rr_get_interval](#)

[sched_setparam](#)

[<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

sched_getparam

This service gets the scheduling parameters of a process specified by pid in to param.

Usage

```
int sched_getparam (pid_t pid,
                    const struct sched_param *param)
```

Arguments

- **pid**
Specifies ID of the process whose parameters are to be get.
- **param**
Pointer to the scheduling parameters of a process.

Return Values

- **POSIX_ERROR**
Indicates that the service fails and sets errno to the appropriate error code.

Example

```
#include "services/pthread.h"
#include "services/sched.h"

pid_t pid;
struct sched_param sp;
int status;

/* currently not supported */
status = sched_getparam(pid,&sp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getscheduler](#)

[sched_setparam](#) <sched.h>

[Nucleus POSIX Threads Library APIs](#)

sched_getscheduler

This service returns the scheduling policy of the process specified by pid.

Usage

```
int sched_getscheduler (pid_t pid)
```

Arguments

- **pid**
Specifies ID of the process whose scheduling policy is to be received.

Return Values

- **POSIX_ERROR**
Indicates that the service fails and sets errno to the appropriate error code.

Example

```
#include "services/pthread.h"
#include "services/sched.h"

int status;
pid_t pid;

/* This function is currently not supported */
status = sched_getscheduler(pid);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[sched_setparam](#) <sched.h>

[Nucleus POSIX Threads Library APIs](#)

sched_rr_get_interval

This service updates the timespec structure referenced by the interval argument to contain the current execution time limit (that is, time quantum) for the process specified by pid.

Usage

```
int sched_rr_get_interval (pid_t      pid,  
                          struct timespec interval)
```

Arguments

- **pid**
Specifies the ID of the process whose time quantum is to be received.
- **interval**
To be filled with the time quantum of the calling process.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:
 - ENOTSUP**
If SCHED_OTHER is given in the policy argument.
 - EINVAL**
The value of the policy parameter does not represent a defined scheduling policy.

Description

Only the current execution time limit for the calling process is supported, as there is only one process in the case of Nucleus POSIX. If pid is set to any value other than 0, the result will be an error. Not pid = 0 is the current default.

Example

```
pid_t pid;  
struct timespec interval;  
  
/* Set the pid to the ID of the calling process */  
pid = 0;  
  
/* Get the time quantum value for the calling process */  
ret = sched_rr_get_interval (pid,&interval);
```


Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[sched_get_priority_max](#) [sched_getscheduler](#)

[sched_setparam](#) [<sched.h>](#)

[Nucleus POSIX Threads Library APIs](#)

sched_setparam

This service sets the scheduling parameters of the process specified by pid.

Usage

```
int sched_setparam (pid_t pid,
                   const struct sched_param *param)
```

Arguments

- pid
Specifies ID of the process whose parameters are to be set.
- param
Pointer to the scheduling parameter structure.

Return Values

- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:

Example

```
#include "services/pthread.h"
#include "services/sched.h"

pid_t pid;
struct sched_param sp;
int status;

/* Set it to FIFO */
sp.sched_priority = SCHED_FIFO;

/* currently not supported */
status = sched_setparam(pid,&sp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sched_getparam](#)

[sched_getscheduler](#) <sched.h>

[Nucleus POSIX Threads Library APIs](#)

sched_yield

This service causes the current thread to yield its execution in favor of another thread with the same or greater priority.

Usage

```
int sched_yield (void)
```

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.

Example

```
int ret;  
  
ret = sched_yield();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <sched.h>

[Nucleus POSIX Threads Library APIs](#)

sem_close

This service closes the named semaphore. This service indicates that the calling thread is finished using the named semaphore indicated by sem.

Usage

```
int sem_close (sem_t *sem)
```

Arguments

- sem
Pointer to the semaphore ID.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
sem points to an illegal address or sem is NULL.

Example

```
sem_t sem;  
int ret;  
  
/* Close an already created named semaphore */  
ret = sem_close(&sem);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_init](#)

[sem_open](#)

[sem_unlink](#)

<semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

sem_destroy

This service destroys any state associated with the unnamed semaphore pointed to by sem. The space for storing the semaphore is not freed.

Usage

```
int sem_destroy (sem_t *sem)
```

Arguments

- sem
Pointer to the semaphore ID.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EINVAL
sem points to an illegal address or sem is NULL.
 - EBUSY
There are currently threads blocked on the semaphore.

Example

```
sem_t sem;  
int ret;  
  
/* Destroy an already created unnamed semaphore */  
ret = sem_destroy(&sem);  
  
This function is not interruptible by the delivery of a signal.
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_init](#)

[sem_open](#) <semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

sem_getvalue

This service updates the location referenced by the sval argument to have the value of the semaphore referenced by sem without affecting the state of the semaphore. It is used both for the named and the unnamed semaphore.

Usage

```
int sem_getvalue (sem_t *sem,
                  int    *sval)
```

Arguments

- sem
Pointer to the semaphore ID.
- sval
Pointer to the semaphore value.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The value of sem or sval is NULL or the semaphore is invalid.

Example

```
sem_t sem;
int sval;
int ret;

/* Get the value of an already created semaphore */
ret = sem_getvalue(&sem, &sval);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_post](#)

[sem_timedwait](#)

[sem_trywait](#)

[sem_wait](#)

[<semaphore.h>](#)

[Nucleus POSIX Threads Library APIs](#)

sem_init

This service initializes an unnamed semaphore pointed to by sem to value amount.

Usage

```
int sem_init (sem_t      *sem,  
             int         pshared,  
             unsigned int value)
```

Arguments

- sem
Pointer to the semaphore to be filled with the ID of the created semaphore.
- pshared
Specifies the flag to indicate whether the semaphore is shared between the processes or not. The pshared parameter must be set to 0. Setting pshared to a nonzero value will trigger the ENOTSUP error.
- value
Specifies the initial value of the semaphore with which the semaphore is to be initialized.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EINVAL
The value argument exceeds SEM_VALUE_MAX or value of sem is NULL.
 - ENOTSUP
If pshared is given a value other than 0. Currently the sharing of the semaphores between the processes is not supported.
 - ENOSPC
A resource required to initialize the semaphore has been exhausted. The limit on semaphores SEM_NSEMS_MAX has been reached.

Description

A semaphore cannot be shared between the processes because, in Nucleus POSIX, there is only one process so pshared must be set to 0.

Multiple threads must not initialize the same semaphore. Also, do not reinitialize a semaphore while other threads might be using it.

Example

```
sem_t sem;
int pshared;
int ret;
int value;

/* Initialize the variables */
pshared = 0;
value   = 1;

/* Create the binary semaphore */
ret = sem_init(&sem, pshared, value);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[sem_post](#)

[sem_trywait](#)

[<semaphore.h>](#)

[sem_destroy](#)

[sem_timedwait](#)

[sem_wait](#)

[Nucleus POSIX Threads Library APIs](#)

sem_open

This service establishes a connection between a named semaphore and a thread.

Usage

```
sem_t* sem_open (char      *name,  
                int       oflag,  
                /* mode_t   mode,  
                 unsigned value*/)
```

Arguments

- name

Specifies the semaphore name to be opened. This name is a simple string and does not appear in the file system.

- oflag

Specifies whether semaphore is going to be accessed or just created.

O_CREAT

Create a semaphore if it does not already exist. If it is set and the semaphore already exists, then O_CREAT has no effect. This also requires a third and a fourth argument: mode, which is of type mode_t, and value, which is of type unsigned.

O_EXCL

If O_EXCL and O_CREAT are set, the service fails if the semaphore name exists.

- mode (optional)

We do not support it as we are not using the file system for semaphores.

- value (optional)

Specifies the value with which the semaphore is to be initialized.

Return Values

- Pointer to the created semaphore on success.
- SEM_FAILED

Indicates that the service fails and sets errno to the corresponding error code:

ENAMETOOLONG

The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

EINVAL

O_CREAT was specified in oflag and value was greater than {SEM_VALUE_MAX}.

EEXIST

O_CREAT and O_EXCL are set and the named semaphore already exists.

ENOENT

O_CREAT is not set and the named semaphore does not exist.

ENOSPC

There is insufficient space for the creation of the newly named semaphore.

Description

The name argument points to a string naming a semaphore object. This implementation treats the semaphore name like a string of characters, without regarding the file system namespace, directories or anything related to these.

Example

```
char name[] = "semaphore1";
int oflag;
mode_t mode;
unsigned value;
sem_t * sem;

oflag = O_CREAT;
mode = 0;
value = 1;

/* Open a binary semaphore */
sem = sem_open(name, oflag, mode, value);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_close](#)

[sem_post](#) [sem_timedwait](#)

[sem_trywait](#) [sem_unlink](#)

[sem_wait](#) <semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

sem_post

This service automatically increments the semaphore pointed to by sem. This service is used both for the named and the unnamed semaphore. When many threads are blocked on the semaphore, one of them will be unblocked.

Usage

```
int sem_post (sem_t *sem)
```

Arguments

- sem
Pointer to the semaphore ID.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code.

Example

```
sem_t sem;  
int ret;  
  
/* Post an already created semaphore */  
ret = sem_post(&sem);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_timedwait](#)

[sem_trywait](#)

[sem_wait](#)

<semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

sem_timedwait

This service locks the semaphore referenced by `sem`. If the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a [sem_post](#) function, this wait will be terminated when the specified timeout expires.

Usage

```
int sem_timedwait (sem_t *sem,  
                  const struct timespec *abs_timeout)
```

Arguments

- `sem`
Pointer to the semaphore ID.
- `abs_timeout`
Pointer to absolute time out value.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:
 - `EINVAL`
sem points to an illegal address or `sem` is `NULL`.
 - `EINVAL`
The thread would have blocked, and the `abs_timeout` parameter specified a nanoseconds field value less than zero or greater than or equal to one billion.
 - `ETIMEDOUT`
The semaphore could not be locked before the specified timeout expired.

Example

```
sem_t sem;  
int ret;  
struct timespec timeout;  
  
/* Set the timeout value equal to 2 seconds. */  
timeout.tv_sec = 2;  
timeout.tv_nsec = 0;  
  
/* Wait on an already created semaphore */  
ret = sem_timedwait(&sem,&timeout);  
  
/* In our implementation this function is not interruptible by the  
   delivery of a signal. */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <semaphore.h>

[sem_post](#)

[sem_trywait](#)

[time](#)

<time.h>

[Nucleus POSIX Threads Library APIs](#)

sem_trywait

This service automatically decrements the count in the semaphore pointed to by `sem` when the count is greater than zero. This function is a non-blocking version of [sem_wait](#) that is it returns immediately if unsuccessful.

Usage

```
int sem_trywait (sem_t *sem)
```

Arguments

- `sem`
Pointer to the semaphore ID.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:
 - `EINVAL`
 `sem` points to an illegal address or `sem` is `NULL`.
 - `EAGAIN`
 The semaphore was already locked, so it cannot be locked immediately.

Example

```
sem_t sem;  
int ret;  
  
/* trywait on an already created semaphore */  
ret = sem_trywait(&sem);  
  
/* This function is not interruptible by the delivery of a signal. */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_post](#)

[sem_timedwait](#) `<semaphore.h>`

[Nucleus POSIX Threads Library APIs](#)

sem_unlink

This service removes the named semaphore named by the string name. This service will not block until all references have been destroyed. It will return immediately.

Usage

```
int sem_unlink (const char *name)
```

Arguments

- name
Specifies the name of the named semaphore to be unlinked.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and set errno to the corresponding error code:
 - EINVAL
sem points to an illegal address or sem is NULL.
 - ENOENT
The named semaphore does not exist.
 - ENAMETOOLONG
The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

Example

```
const char* name = "semaphore1"
int ret;

/* Unlink an already created named semaphore */
ret = sem_unlink(name);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_close](#)

[sem_open](#) <semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

sem_wait

This service blocks the calling thread until the count in the semaphore pointed to by sem becomes greater than zero, then atomically decrement it.

Usage

```
int sem_wait (sem_t *sem)
```

Arguments

- sem
Pointer to the semaphore ID.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
sem points to an illegal address or sem is NULL.

Example

```
sem_t sem;  
int ret;  
  
/* Wait on an already created semaphore */  
ret = sem_wait(&sem);  
  
/* In our implementation this function is not interruptible by the  
   delivery of a signal. */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_post](#)

[sem_timedwait](#) <semaphore.h>

[Nucleus POSIX Threads Library APIs](#)

setjmp

This service saves the calling environment in its env argument for later use by [longjmp](#).

Usage

```
int setjmp (jmp_buf env)
```

Arguments

- env
Jump array buffer contains all the saved values.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- non-zero
If the return is from a call to [longjmp](#), it returns a non-zero value.

Example

```
#include "services/stdio.h"
#include "services/stdlib.h"
#include "services/setjmp.h"

void one_time(void);
void exits(void);
jmp_buf place;

main()
{
    /* First call returns 0, longjmp() will return 4. */
    if (setjmp(place) != 0)
    {
        printf("longjmp success \n");
        exits();
    }

    one_time();

    /* This statement will never reached - longjmp will go back above. */
    printf("Error, something is wrong!\n");
}

void one_time(void)
{
    /* Return to setjmp, returning 4! */
    longjmp(place, 4);
    printf("Error, something is wrong!\n");
}

void exits(void)
{
    /* forever loop */
}
```

```
        while (1) ;  
    }
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [longjmp](#)

<setjmp.h> [sigsetjmp](#)

[Nucleus POSIX Threads Library APIs](#)

sigaction

This function allows the calling thread to examine and/or specify the action to be associated with a specific signal.

Usage

```
int sigaction (int          sig,
               const struct sigaction *act,
               struct sigaction *oact)
```

Arguments

- **sig**
Specifies the signal number.
- **act**
Action to be associated with the specified signal.
- **oact**
Action previously associated with the signal.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:

EINVAL

The `sig` argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

EINVAL

An attempt was made to set the action to `SIG_DFL` for a signal that cannot be caught or ignored (or both).

Example

```
extern async_handler(int signo, siginfo_t *info, void *context)

struct sigaction act;
struct sigaction oact;
int    status;

/* Fill the act structure */
act.sa_sigaction = async_handler;
act.sa_flags     = SA_SIGINFO;
act.sa_mask      = 0;

/* Set the signal action */
status = sigaction(SIGRTMIN, &act, &oact);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	kill
longjmp	raise
sem_init	sem_open
sigaddset	sigdelset
sigemptyset	sigfillset
sigismember	<signal.h>
sigprocmask	sigsuspend
Nucleus POSIX Threads Library APIs	

sigaddset

This function adds an individual signal (signo) to the signal set pointed to by set.

Usage

```
int sigaddset (sigset_t *set,  
              int      signo)
```

Arguments

- set
Pointer to the signal set.
- signo
Signal number to be added.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The value of the signo argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int      status;  
  
/* Empty the signal set */  
status = sigemptyset(&set);  
  
/* Add SIGRTMIN in the set */  
status = sigaddset(&set, SIGRTMIN);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	sigaction
sigdelset	sigemptyset
sigfillset	sigismember
<signal.h>	sigpending
sigprocmask	sigsuspend

[Nucleus POSIX Threads Library APIs](#)

sigdelset

This function deletes an individual signal (signo) from the signal set pointed to by set.

Usage

```
int sigdelset (sigset_t* set,  
              int      signo)
```

Arguments

- set
Pointer to the set.
- signo
Signal number to be removed.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The value of the signo argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int      status;  
  
/* Fill the signal set */  
status = sigfillset(&set);  
  
/* Delete SIGRTMIN from the set */  
status = sigdelset(&set, SIGRTMIN);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	sigaction
sigaddset	sigemptyset
sigfillset	sigismember
<signal.h>	sigpending
sigprocmask	sigsuspend

[Nucleus POSIX Threads Library APIs](#)

sigemptyset

This function empties the signal set pointed by set.

Usage

```
int sigemptyset (sigset_t *set)
```

Arguments

- set
Pointer to the set.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The value of the signo argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int status;  
  
/* Empty the signal set */  
status = sigemptyset(&set);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sigaction](#)

sigaddset	sigdelset
sigfillset	sigismember
<signal.h>	sigpending
sigprocmask	sigsuspend

[Nucleus POSIX Threads Library APIs](#)

sigfillset

This function enables all the signals in the given set.

Usage

```
int sigfillset (sigset_t *set)
```

Arguments

- `set`
Pointer to the set.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the following error code:
`EINVAL`
The value of the `signo` argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int status;  
  
/* Fill the signal set */  
status = sigfillset(&set);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	sigaction
sigaddset	sigdelset
sigemptyset	sigismember
<signal.h>	sigpending
sigprocmask	sigsuspend

[Nucleus POSIX Threads Library APIs](#)

sigismember

This function tests whether the signal specified by `signo` is a member of the set pointed to by `set`.

Usage

```
int sigismember (const sigset_t *set,  
                int          signo)
```

Arguments

- `set`
Pointer to the set.
- `signo`
Signal number.

Return Values

- `POSIX_FOUND`
If the specified signal is a member of the specified set.
- `POSIX_NOT_FOUND`
If it is not a member of the specified set.
- `POSIX_ERROR`
Indicates that the service has failed and sets `errno` to the corresponding error code.
`EINVAL`
The value of the `signo` argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int status;  
  
/* Fill the signal set */  
status = sigfillset(&set);  
  
/* Check whether SIGRTMIN is a member of set */  
status = sigismember(&set, SIGRTMIN);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	sigaction
sigaddset	sigdelset
sigemptyset	sigfillset
<signal.h>	sigpending
sigprocmask	sigsuspend

Nucleus POSIX Threads Library APIs

siglongjmp

This function restores the environment saved by the most recent invocation of [sigsetjmp](#) in the same thread, with the corresponding `jmp_buf` argument. The `siglongjmp` function restores the saved signal mask if the `env` argument was initialized by a call to [sigsetjmp](#) with a non-zero `savemask` argument.

Usage

```
void siglongjmp (sigjmp_buf env,  
                int      val)
```

Arguments

- `env`
Jump array buffer contains all the saved values.
- `val`
if `val` is 0, `setjmp()` returns 1.

Example

```
#include "services/signal.h"  
#include "services/setjmp.h"  
#include "services/unistd.h"  
#include "services/stdio.h"  
  
int main()  
{  
    sigset_t sigset;  
    sigjmp_buf mark;  
    sigemptyset( &sigset );  
    sigaddset( &sigset, SIGUSR1 );  
    sigaddset( &sigset, SIGUSR2 );  
    sigprocmask( SIG_SETMASK, &sigset, NULL );  
  
    /* the SIGUSR1 and SIGUSR2 signals will be saved as part of the  
       environment by the sigsetjmp() function. */  
    if ( sigsetjmp( mark, 1 ) != 0 )  
    {  
        /* the sigset are recovered to original settings */  
        printf( "return from siglongjmp()\n" );  
  
        sigprocmask( SIG_SETMASK, NULL, &sigset );  
        if ( sigismember( &sigset, SIGUSR1 ) )  
            printf( "signal mask restored after siglongjmp() \n" );  
        printf( "sigsetjmp(), siglongjmp() operations are succesfull. \n");  
    }  
    else  
    {  
        printf( "sigsetjmp() has been called\n" );  
  
        /* the SIGUSR1 and SIGUSR2 signals are deleted from sigset */  
        sigdelset( &sigset, SIGUSR1 );  
        sigdelset( &sigset, SIGUSR2 );  
        sigprocmask( SIG_SETMASK, &sigset, NULL );  
    }  
}
```

```
        siglongjmp( mark, -1 );
        printf("sigsetjmp(), siglongjmp() operations error. \n");
    }
    return(0);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [longjmp](#)

[setjmp](#) [<setjmp.h>](#)

[sigprocmask](#) [sigsetjmp](#)

[sigsuspend](#) [Nucleus POSIX Threads Library APIs](#)¹

sigpending

This function stores, in the location referenced by the set argument, the set of signals that are blocked from delivery to the calling thread.

Usage

```
int sigpending (sigset_t *set)
```

Arguments

- set
Pointer to the set.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The value of the signo argument is an invalid or unsupported signal number.

Example

```
sigset_t set;  
int status;  
  
/* Get the pending signal set */  
status = sigpending(&set);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sigaddset](#)
[sigdelset](#) [sigemptyset](#)
[sigfillset](#) [sigismember](#)
[sigprocmask](#) [<signal.h>](#)
[Nucleus POSIX Threads Library APIs](#)

sigprocmask

This function examines or changes (or both) the signal mask of the calling thread.

Note



This API is very important for enabling signals within the current thread.

Usage

```
int sigprocmask (int          how,  
                const sigset_t *set,  
                sigset_t      *oset)
```

Arguments

- **how**

Indicates the way the set is changed. This can be set to one of the following values:

SIG_BLOCK

Union of the current set and the signal set pointed out by set.

SIG_SETMASK

Signal set pointed out by set.

SIG_UNBLOCK

Intersection of the current set and the complement of the signal set pointed out by set.

- **set**

Pointer to the given input set.

- **oset**

Pointer to the output old set.

Return Values

- **POSIX_SUCCESS**

Indicates successful completion of the service.

- **POSIX_ERROR**

Indicates that the service fails and sets `errno` to the following error code:

EINVAL

The value of the `how` argument is not equal to one of the defined values.

Example

```
sigset_t oset;  
sigset_t set;  
int status;  
  
/* Empty the signal set */  
sigemptyset(&set);
```

```
/* Add SIGRTMIN in the set */
sigaddset(&set, SIGRTMIN);

/* Set the calling thread's signal mask */
status = sigprocmask(SIG_SETMASK, &set, &oset);

/* The action will enable signals for the current thread */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sigaction](#)

[sigaddset](#)

[sigdelset](#)

[sigemptyset](#)

[sigfillset](#)

[sigismember](#)

[<signal.h>](#)

[sigpending](#)

[sigqueue](#)

[sigsuspend](#)

[Nucleus POSIX Threads Library APIs](#)

sigqueue

This function queues the signal to a thread.

Usage

```
int sigqueue (pid_t          pid,  
              int            signo,  
              const union sigval value)
```

Arguments

- **pid**
Process ID (Thread ID).
- **signo**
Signal number to be sent.
- **value**
Signal value.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:
 - EINVAL**
The value of the `signo` argument is an invalid or unsupported signal number.
 - EAGAIN**
No resources are available to queue the signal. The thread has already `SIGQUEUE_MAX` signals that are still pending at the receiver.
 - ESRCH**
The process `pid` (`tid`) does not exist.

Example

```
/* current implementation the user must teardown File system or Networking  
   connection manually. */  
  
int      sock1;  
pthread_t Thread_Server;  
  
/* Server thread accepting connection */  
void* thread_server_entry(void *argv)  
{  
    struct sockaddr_in serv_addr;  
    socklen_t len;  
    struct sigaction act;
```



```
sigset_t set;
buffer[50];
sigset_t save_set;

/* SIGUSR1 is by default pre-set to kill thread, we don't want this, we
   want to re-wire SIGUSR1 to teardown the current TCP connection first
   then kill this thread. */
act.sa_sigaction = signal_handler;
act.sa_flags      = 0;
act.sa_mask       = 0;
sigaction(SIGUSR1, &act, NULL);

/* Let SIGUSR1 through, Now the user have SIGUSR1, SIGKILL, SIGSTOP and
   SIGCONT signals available to get through to interrupt this thread.*/
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
pthread_sigmask(SIG_UNBLOCK, &set, &save_set);

/* Assuming the user has perform all the necessary steps of creating
   socket, bind and listen for TCP connection. */
...

/* An example where the server thread is hang forever and need to be
   destroyed! The read is interrupted and signal handler is invoked. */
sts = read(sock1, buffer, 50);

...
}

void* thread_client_entry(void *argv)
{
    /* Signal thread_server_entry. SIGUSR1 has been configured to invoke
       signal_handler().
       Important: Keep in mind that multiple signals calls to the
       Thread_Server will be queued in the order the signal is received. */
    sigqueue(Task_Server1, SIGUSR1, sock1);

    /* Give below signal catching function
       chance to run. */
    sleep(1);
}

/* Signal handler for tearing down TCP connection */
void signal_handler(int signo)
{
    /* shutdown() can be be used to tear down socket recv() or
       send() connection */

    /* close the socket, it is also possible to close read(),
       write(), recv(), send(), recvfrom(), etc. */

    if (shutdown(sock1, SHUT_RD) == -1)
    {
        printf("error\n");
    }
}
```

```
/* Perform any other necessary cleanup */  
...  
  
/* Destroy the server thread all together after done all the necessary  
   cleanup. If pthread_exit() is not invoked, the user will continue  
   execution of thread_server_entry(). */  
pthread_exit(NULL);  
}
```

Caution



With the current implementation, It is very important that you close the socket or file descriptor when signaling a suspended networking or file system operation.

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <signal.h>

[Nucleus POSIX Threads Library APIs](#)

sigsetjmp

This service saves the calling environment in its env argument for later use by [siglongjmp](#).

Usage

```
int sigsetjmp (sigjmp_buf env,  
              int      savemask)
```

Arguments

- env
Jump array buffer contains all the saved values.
- savemask
If the value of the savemask argument is not 0, sigsetjmp() saves the current signal mask of the calling thread as part of the calling environment.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- non-zero
If the return is from a call to [siglongjmp](#)(), it returns a non-zero value.

Example

```
#include "services/signal.h"  
#include "services/setjmp.h"  
#include "services/unistd.h"  
#include "services/stdio.h"  
  
int main()  
{  
    sigset_t sigset;  
    sigjmp_buf mark;  
    sigemptyset( &sigset );  
    sigaddset( &sigset, SIGUSR1 );  
    sigaddset( &sigset, SIGUSR2 );  
    sigprocmask( SIG_SETMASK, &sigset, NULL );  
  
    /* the SIGUSR1 and SIGUSR2 signals will be saved as part of the  
       environment by the sigsetjmp() function. */  
    if ( sigsetjmp( mark, 1 ) != 0 )  
    {  
        /* the sigset are recovered to original settings */  
        printf( "return from siglongjmp()\n" );  
  
        sigprocmask( SIG_SETMASK, NULL, &sigset );  
  
        if ( sigismember( &sigset, SIGUSR1 ) )  
            printf( "signal mask restored after siglongjmp() \n" );  
        printf( "sigsetjmp(), siglongjmp operations are successful. \n");  
    }  
}
```

```
else
{
    printf( "sigsetjmp() has been called\n" );

    /* the SIGUSR1 and SIGUSR2 signals are deleted from sigset */
    sigdelset( &sigset, SIGUSR1 );
    sigdelset( &sigset, SIGUSR2 );

    sigprocmask( SIG_SETMASK, &sigset, NULL );
    siglongjmp( mark, -1 );
    printf("sigsetjmp(), siglongjmp() operations error. \n");
}
return(0);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<setjmp.h>](#)

[siglongjmp](#)

[sigprocmask](#)

[sigsuspend](#)

[Nucleus POSIX Threads Library APIs](#)

sigsuspend

This function replaces the current signal mask of the calling thread with the set of signals pointed to by sigmask and then suspends the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the thread.

Usage

```
int sigsuspend (const sigset_t* sigmask)
```

Arguments

- sigmask
Pointer to the signal mask.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:

EINTR

A signal is caught from the calling thread and control is returned from the signal-catching function.

Example

```
int status;

/* You don't need to call pthread_sigmask() because it will be invoked
within sigsuspend() API unlike sigwait(), sigtimedwait() and
sigwaitinfo() APIs. Here is how sigsuspend() signal being enabled
within sigsuspend() API: sigset_t sigmask = NULL; with the example of
sigsuspend(NULL) mask = ~sigmask; Replace the current signal mask of
the calling thread with the set of signals pointed to by sigmask. */

/* Suspend until any of the signal arrives */
status = sigsuspend(NULL);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[sigaddset](#)

[sigemptyset](#)

[<signal.h>](#)

[sigaction](#)

[sigdelset](#)

[sigfillset](#)

[Nucleus POSIX Threads Library APIs](#)

sigtimedwait

This function waits for the time interval specified in the timespec structure referenced by timeout, if none of the signals specified by set are pending.

Usage

```
int sigtimedwait (const sigset_t      *set,  
                  siginfo_t          *info,  
                  const struct timespec *timeout)
```

Arguments

- **set**
Pointer to the given set.
- **info**
Pointer to the info to be filled.
- **timeout**
Time to wait.

Return Values

- The signal number found.
- **POSIX_ERROR**

Indicates that the service fails and sets errno to the following error code:

EINVAL

The timeout argument specified a tv_nsec value less than zero or greater than or equal to one billion.

Example

```
sigset_t set;  
siginfo_t info;  
struct timespec timeout;  
int status;  
  
/* Empty the signal set */  
sigemptyset(&set);  
  
/* Add SIGRTMIN in the signal set */  
sigaddset(&set, SIGRTMIN);  
  
/* Set the timeout of 1 second */  
timeout.tv_sec = 1;  
timeout.tv_nsec = 0;  
  
/* sigprocmask and pthread_sigmask are very important APIs for  
   enabling signals. You must call sigprocmask or pthread_sigmask  
   to enable the signals for this thread. */  
pthread_sigmask(SIG_UNBLOCK, &set, &save_set);
```

```
/* Wait for the signal SIGRTMIN */  
status = sigtimedwait(&set,&info,&timeout);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_sigmask](#)

[<signal.h>](#)

[sigsuspend](#)

[<time.h>](#)

[pause](#)

[sigaction](#)

[sigpending](#)

[sigwait](#)

[Nucleus POSIX Threads Library APIs](#)

sigwait

This function selects a pending signal from set, atomically clears it from the system's set of pending signals, and returns that signal number in the location referenced by sig. If no signal in set is pending at the time of the call, the thread is suspended until one or more becomes pending.

Usage

```
int sigwait (const sigset_t *set,
             int             *sig)
```

Arguments

- set
Pointer to the given set.
- sig
Pointer to be filled with the signal number of the received signal.

Return Values

- POSIX_SUCCESS
If the call is successful.
- POSIX_ERROR
Indicates that the service fails and sets errno to the following error code:
EINVAL
The set argument contains an invalid or unsupported signal number.

Example

```
sigset_t set;
int      sig;
int      status;

/* Empty the signal set */
sigemptyset(&set);

/* Add SIGRTMIN in the signal set */
sigaddset(&set, SIGRTMIN);

/* sigprocmask and pthread_sigmask are very important APIs for
   enabling signals. You must call sigprocmask or pthread_sigmask
   to enable the signals for this thread. */
status = sigwait(&set, &sig);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[pthread_sigmask](#)

[pause](#)

[sigaction](#)

<signal.h>

[sigsuspend](#)

<time.h>

[sigpending](#)

[sigwaitinfo](#)

[Nucleus POSIX Threads Library APIs](#)

sigwaitinfo

This function causes the calling thread to be suspended until the signal intended for this thread is caught.

Usage

```
int sigwaitinfo (const sigset_t *set,
                 siginfo_t      *info)
```

Arguments

- `set`
Pointer to the given set.
- `info`
Pointer to the info to be filled.

Return Values

- The signal number found.
- `POSIX_ERROR`

Indicates that the service fails and sets `errno` to the following error code:

`EINTR`

A signal is caught from the calling thread and control is returned from the signal-catching function.

Example

```
sigset_t  set;
siginfo_t info;
int       status;

/* Empty the signal set */
sigemptyset(&set);

/* Add SIGRTMIN in the signal set */
sigaddset(&set, SIGRTMIN);

/* sigprocmask and pthread_sigmask are very important APIs for
   enabling signals. You must call sigprocmask or pthread_sigmask
   to enable the signals for this thread. */
pthread_sigmask(SIG_UNBLOCK, &set, &save_set);
status = sigwaitinfo(&set, &info);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [pause](#)

[pthread_sigmask](#)

[sigaction](#)

<signal.h>

[sigsuspend](#)

<time.h>

[sigpending](#)

[sigwait](#)

[Nucleus POSIX Threads Library APIs](#)

sleep

This service causes the calling thread to be suspended from execution until either the number of real time seconds specified by the argument `seconds` has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal catching function. The suspension time may be longer than requested due to the scheduling of other activities by the system.

Usage

```
unsigned sleep (unsigned seconds)
```

Arguments

- `seconds`
Time in seconds to sleep.

Return Values

- 0
The requested time has elapsed.
- If the signal has been delivered, this function returns the "unslept" amount (the requested time minus the time actually slept) in seconds.

Example

```
/* This example illustrates how to interrupt a lazy thread
   that is sleeping for long period of time. The lazy thread
   will invoke a signal handler and the sleep() API will
   return the amount of time remaining in the sleep. */

pthread_t lazy_thread_id = 0;
int sig_counter          = 0;

/* Asuming another thread spawns lazy_thread with thread id
   lazy_thread_id */

...

/* Simple handler to illustrate sleep gets interrupted in the middle of
   sleep. */
void simple_signal_handler(int signo)
{
    sig_counter++;
}

/* Thread 0 will be interrupted before it's done sleeping for 15
   seconds */
void* lazy_thread(void* argv)
{
    pthread_attr_t attr;
    pthread_t wakeup_id;
    struct sigaction act;
    sigset_t set;
    sigset_t save_set;
    int unslept;
```

```
/* Let SIGUSR2 through */
sigemptyset(&set);
sigaddset(&set, SIGUSR2);
pthread_sigmask(SIG_UNBLOCK, &set, &save_set);

/* Re-wire SIGUSR2 to invoke simple_signal_handler. */
act.sa_handler = simple_signal_handler;
act.sa_flags = 0;
act.sa_mask = 0;
sigaction(SIGUSR2, &act, NULL);

/* Initialize thread attribute for this thread.
   This thread is by default created as PTHREAD_CREATE_JOINABLE. */
pthread_attr_init(&attr);

/* Create a thread that will wake me up in the middle of sleeping. */
if (pthread_create(&wake_up_id, &attr, wake_up_thread, NULL) != 0)
{
    printf("error\n");
}

/* Let wake_up_thread run....
   wake_up_thread suppose to wake me up before 15 seconds is up. */
unslept = sleep(15);

/* I'm here after the signal handler. Keep in mind that POSIX signal is
   queued unlike Nucleus PLUS so sending signals multiple times will
   result counter bigger than 1. */
if (sig_counter == 1)
    printf("Signal handler is succesfully invoked once!");

/* print the remaining time */
printf("I'm only sleeping for %i seconds!", (int)(15-unslept));

/* I'm detaching this thread so that Nucleus POSIX can delete this
   thread and recycle the resources. Keep in mind, this thread is no
   longer PTHREAD_CREATE_JOINABLE! This is the same as thread created
   with detachable attribute
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
   or pthread_exit(NULL). */
pthread_detach(lazy_thread_id);
}

/* I will wake up lazy_thread. */
void* wake_up_thread(void *argv)
{
    /* Interrupting lazy_thread */
    pthread_kill(Task_0, SIGUSR2);

    /* Give the scheduler chance to switch thread back to lazy_thread. */
    sleep(1);

    /* Let Nucleus POSIX recycle the resources */
    pthread_exit(NULL);
}
```

Related Topics

System Interfaces volume of IEEE Std
1003.1-2001

[sleep](#)

[Nucleus POSIX Threads Library APIs](#)

timer_create

This service creates a timer using the specified clock.

Usage

```
int timer_create (clockid_t      clock_id,  
                 struct sigevent *evp,  
                 timer_t        *timerid)
```

Arguments

- clock_id

Specifies ID of the clock whose time is to be set. The clock_id parameter can be set to one of the following:

CLOCK_REALTIME

Identifier for the system wide real-time clock.

- evp

Pointer to the timespec structure, to be filled with the resolution of the specified clock.

- timerid

Pointer to the ID of the timer created.

Return Values

- POSIX_SUCCESS

Indicates successful completion of the service.

- POSIX_ERROR

Indicates that the service fails and sets errno to the corresponding error code:

EINVAL

The clock_id argument does not specify a known clock.

EAGAIN

There are not sufficient resources available to satisfy the request.

EINVAL

If the evp is invalid.

Example

```
clockid_t clockid;  
struct sigevent evp;  
timer_t timerid;  
int ret;  
  
clockid = CLOCK_REALTIME;  
  
evp.sigev_notify = SIGEV_SIGNAL;  
evp.sigev_signo = SIGALRM;
```

```
evp.sigev_value.sival_int = 0;
evp.sigev_value.sival_ptr = NULL;

/* Create a timer */
ret = timer_create(clockid, &evp, &timerid);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [clock_getres](#)

[timer_delete](#) <time.h>

[Nucleus POSIX Threads Library APIs](#)

timer_delete

This service deletes the specified timer.

Usage

```
int timer_delete (timer_t timerid)
```

Arguments

- **timerid**
Specifies the ID of the timer to be deleted.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the following error code:
EINVAL
The timer ID specified by `timerid` is not a valid timer ID.

Example

```
timer_t timerid;  
int ret;  
  
/* Delete an already created timer of ID timerid. */  
ret = timer_delete(timerid);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [timer_create](#)

<time.h>

[Nucleus POSIX Threads Library APIs](#)

timer_gettime

This service gets the amount of time until the specified timer expires.

Usage

```
int timer_gettime (timer_t      timerid,  
                  struct itimerspec *value)
```

Arguments

- **timerid**
Specifies the ID of the timer whose time is to be get.
- **value**
To be filled with the amount of time before the timer expires.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:
 - EINVAL**
The timer ID specified by `timerid` is not a valid timer ID.
 - EINVAL**
The value is `NULL`.

Example

```
timer_t timerid;  
struct itimerspec value;  
int ret;  
  
/* Get the remaining time from the already created timer before the timer  
   expires */  
ret = timer_gettime(timerid,&value);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [clock_getres](#)

[timer_create](#) [<time.h>](#)

[Nucleus POSIX Threads Library APIs](#)

timer_settime

This service sets the time until the next expiration of the timer specified by timerid from the it_value member of the value argument and arm the timer if the it_value member of value is non-zero.

Usage

```
int timer_settime (timer_t          timerid,  
                  int              flags,  
                  const struct itimerspec *value,  
                  struct itimerspec  *ovalue)
```

Arguments

- timerid
Specifies the ID of the timer to be deleted.
- flags
Specifies flags to govern setting the time.
TIMER_ABSTIME
Indicates time is absolute.
- value
Specifies the initial time and the reschedule time to be set.
- ovalue
To be filled with the old value of time.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
EINVAL
The timer ID specified by timerid is not a valid timer ID.
EINVAL
A value structure specified a nanosecond value less than zero or greater than or equal to 1 000 million, and the it_value member of that structure did not specify zero seconds and nanoseconds.
EINVAL
The value is NULL.

Example

```
timer_t timerid;
int flags;
struct itimerspec value;
struct itimerspec ovalue;
int ret;

flags = TIMER_ABSTIME

val.it_value.tv_sec = 4;
val.it_value.tv_nsec = 0;

/* Non Periodic Timer */
val.it_interval.tv_sec = 0;
val.it_interval.tv_nsec = 0;

/* Set the timer to four seconds */.
ret = timer_settime(timerid, flags, &value, &ovalue);
```

Related Topics


Base Definitions volume of IEEE Std 1003.1-2001 [clock_getres](#)

[timer_create](#) [<time.h>](#)

[Nucleus POSIX Threads Library APIs](#)

uname

This function stores information identifying the current system in the structure pointed to by name.

 **Note** Machine name and node name are left empty.

Usage

```
int uname (struct utsname *name)
```

Arguments

- name
Pointer to the utsname structure.

Return Values

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code.

Example

```
#include "services/sys/utsname.h"

struct utsname name;
int ret;

/* Get the information of the system */
ret = uname(&name);
printf("System name : %s \n",name.sysname);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <sys/utsname.h>
2001

[Nucleus POSIX Threads Library APIs](#)

Chapter 3

Nucleus POSIX Run Time Library APIs

This chapter provides a detailed reference of all the Nucleus POSIX Run Time Library (POSIX RTL) APIs.

- [abort](#)
- [abs](#)
- [acos](#)
- [Alarm](#)
- [asctime](#)
- [asctime_r](#)
- [asin](#)
- [atan](#)
- [atan2](#)
- [atof](#)
- [atoi](#)
- [atol](#)
- [atoll](#)
- [bcmp](#)
- [bcopy](#)
- [bsearch](#)
- [bzero](#)
- [calloc](#)
- [ceil](#)
- [clearerr](#)
- [clock](#)
- [clock_nanosleep](#)
- [confstr](#)

- [cos](#)
- [cosh](#)
- [ctime](#)
- [ctime_r](#)
- [difftime](#)
- [div](#)
- [exp](#)
- [fabs](#)
- [fclose](#)
- [fdopen](#)
- [feclearexcept](#)
- [fegetenv](#)
- [fegetexceptflag](#)
- [fegetround](#)
- [feholdexcept](#)
- [feof](#)
- [feraiseexcept](#)
- [ferror](#)
- [fesetenv](#)
- [fesetexceptflag](#)
- [fesetround](#)
- [fetestexcept](#)
- [feupdateenv](#)
- [fflush](#)
- [ffs](#)
- [fgetc](#)
- [fgetpos](#)
- [fgets](#)
- [fileno](#)

-
- flockfile
 - floor
 - fmod
 - fopen
 - fpathconf
 - fprintf
 - fputc
 - fputs
 - fread
 - free
 - frexp
 - freopen
 - fscanf
 - fseek
 - fsetpos
 - ftell
 - frylockfile
 - funlockfile
 - fwrite
 - getc
 - getc_unlocked
 - getchar
 - getchar_unlocked
 - getenv
 - getpid
 - gets
 - gmtime
 - gmtime_r
 - hypot

- [imaxabs](#)
- [imaxdiv](#)
- [isalnum](#)
- [isalpha](#)
- [isascii](#)
- [isblank](#)
- [isctrl](#)
- [isdigit](#)
- [isgraph](#)
- [islower](#)
- [isprint](#)
- [ispunct](#)
- [isspace](#)
- [isupper](#)
- [isxdigit](#)
- [labs](#)
- [ldexp](#)
- [ldiv](#)
- [llabs](#)
- [lldiv](#)
- [localtime](#)
- [localtime_r](#)
- [log](#)
- [log10](#)
- [malloc](#)
- [memchr](#)
- [memcmp](#)
- [memcpy](#)
- [memmove](#)

-
- [memset](#)
 - [mktime](#)
 - [modf](#)
 - [mq_timedsend](#)
 - [pathconf](#)
 - [perror](#)
 - [pow](#)
 - [printf](#)
 - [pthread_atfork](#)
 - [pthread_getcpuclockid](#)
 - [pthread_mutex_timedlock](#)
 - [pthread_mutexattr_gettype](#)
 - [pthread_mutexattr_settype](#)
 - [putc](#)
 - [putc_unlocked](#)
 - [putchar](#)
 - [putchar_unlocked](#)
 - [puts](#)
 - [qsort](#)
 - [rand](#)
 - [rand_r](#)
 - [realloc](#)
 - [remove](#)
 - [rewind](#)
 - [scanf](#)
 - [setbuf](#)
 - [setenv](#)
 - [setlocale](#)
 - [setvbuf](#)

- [signal](#)
- [sin](#)
- [sinh](#)
- [snprintf](#)
- [sprintf](#)
- [sqrt](#)
- [srand](#)
- [sscanf](#)
- [strcat](#)
- [strchr](#)
- [strcmp](#)
- [strcoll](#)
- [strcpy](#)
- [strcspn](#)
- [strdup](#)
- [strerror](#)
- [strerror_r](#)
- [strftime](#)
- [strlen](#)
- [strncat](#)
- [strncmp](#)
- [strncpy](#)
- [strpbrk](#)
- [strrchr](#)
- [strspn](#)
- [strstr](#)
- [strtod](#)
- [strtof](#)
- [strtoimax](#)

- [strtok](#)
- [strtok_r](#)
- [strtol](#)
- [strtoll](#)
- [strtold](#)
- [strtoul](#)
- [strtoull](#)
- [strtoumax](#)
- [strxfrm](#)
- [swab](#)
- [sysconf](#)
- [tan](#)
- [tanh](#)
- [time](#)
- [tmpfile](#)
- [tmpnam](#)
- [toascii](#)
- [tolower](#)
- [toupper](#)
- [ungetc](#)
- [unsetenv](#)
- [vfprintf](#)
- [vsnprintf](#)

abort

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function causes abnormal process termination to occur unless the signal SIGABRT is being caught and the signal handler does not return.

Usage

```
void abort (void)
```

Example

```
/* Abnormal process termination to occur */  
abort();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [raise](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

abs

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the absolute value of its integer operand, i.

Usage

```
int abs (int i)
```

Arguments

- i
Integer operand.

Return Value

- int
Absolute value of the integer operand.

Example

```
#include "services/stdlib.h"

static int target, result;
...
target = 65;
...
result = abs(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fabs](#)
[labs](#) [<stdlib.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

acos

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the principal value of the arc cosines of their argument x. The value of x should be in the [1,1] range.

Usage

```
double acos (double x)
```

Arguments

- x
The argument converted to arc cosine.

Return Value

- double
Arc cosine of x will be returned in the [0,] radians range upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 1;
...
result = acos(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [cos](#)

[<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

Alarm

This function schedules an alarm signal.

Nucleus POSIX System side alarm affects only the calling thread and not the whole system (thread-wise signal). This is different than the user side alarm, which affects the entire process (process-wise signal).

Usage

```
unsigned alarm(unsigned seconds)
```

Arguments

- seconds

The number of real-time seconds that elapse before the generation of a SIGALRM signal for the process. If seconds is zero, a pending alarm request, if any, is canceled.

Return Values

- 0

The service has completed successfully.

- nonzero

A previous alarm request has time remaining. This function returns a non-zero value that is the number of seconds until the previous request would have generated a SIGALRM signal.

Example

```
#include "services/pthread.h"
#include "services/unistd.h"
#include "services/time.h"
#include "services/signal.h"
#include "services/stdio.h"

/* Global Variables */
static int Posix_Error      = 1;
static int Psx_Alarm_Counter = 0;
static void TESTS_POSIX_Alarm_Set_Signal(int sig);
static void Test_Psx_Alarm_Handler(int sig);

/* The signal handler will increment the alarm counter. */
static void Test_Psx_Alarm_Handler(int sig)
{
    Psx_Alarm_Counter++;
}

/* Set the signal mask for the current thread */
static void TESTS_POSIX_Alarm_Set_Signal(int sig)
{
    sigset_t signal_mask;

    /* Let sig signal get through */
    sigemptyset(&signal_mask);
    sigaddset(&signal_mask, sig);
}
```

```
    pthread_sigmask(SIG_UNBLOCK, &signal_mask, NULL);
}

/* Perform the test of replacing one alarm after another and
   sleeping long enough for the alarm to occur. This thread will
   perform self cleanup on its own. */

void *Posix_Main_Entry(int argc, void *argv)
{
    int status;

    /* Set SIGALRM mask for this thread */
    TESTS_POSIX_Alarm_Set_Signal(SIGALRM);

    /* Replacing default SIGALRM with new signal value.
       Otherwise default System SIGALRM handler will be used. */
    if (signal(SIGALRM, Test_Psx_Alarm_Handler) != SIG_ERR)
    {
        /* Set the alarm for 1 second */
        status = alarm(1);
        if (status == 0)
        {
            /* Set the alarm for 3 second */
            status = alarm(3);
            if (status > 0)
            {
                /* Sleep long enough to let alarm handler get executed. */
                sleep(5);

                /* Make sure the counter is not set to 1 */
                if (Psx_Alarm_Counter != 1)
                {
                    printf("error error count: %i,
                           line: %i \n", Posix_Error++, __LINE__);
                }
            }
            else
            {
                printf("error error count: %i, line: %i \n",
                       Posix_Error++, __LINE__);
            }
        }
        else
        {
            printf("error error count: %i, line: %i \n",
                   Posix_Error++, __LINE__);
        }
    }
    ...
}
```

Related Topics

[alarm](#)

[pause](#)

[sigaction](#)

[sleep](#)

the Base Definitions volume of IEEE Std
1003.1-2001

<signal.h>

<unistd.h>

[Nucleus POSIX Run Time Library APIs](#)

asctime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function converts the broken-down time in the structure pointed to by `timeptr` into a string in the form: `Sun Sep 16 01:03:52 1973\n\0`. This function is non-reentrant.

Usage

```
char* asctime (const struct tm *timeptr)
```

Arguments

- `timeptr`
Pointer to the `tm` structure

Return Value

- `char*`
Pointer to the converted time if successful.
- `NULL`
Not successful.

Example

```
#include "services/time.h"

char *convtime;
struct tm time_now;

time_now.tm_wday = 03;
time_now.tm_mon  = 07;
time_now.tm_mday = 21;
time_now.tm_hour = 05;
time_now.tm_min  = 59;
time_now.tm_sec  = 23;
time_now.tm_year = 73;

/* Convert the time into asctime */
convtime = asctime(&time_now);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[ctime](#)

[gmtime](#)

[mktime](#)

[time](#)

[<time.h>](#)

[clock](#)

[difftime](#)

[localtime](#)

[strftime](#)

[utime](#)

[Nucleus POSIX Run Time Library APIs](#)

asctime_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function converts the broken-down time in the structure pointed to by `timeptr` into a string in the form: `Sun Sep 16 01:03:52 1973\n\0`. This is the re-entrant version of [asctime](#).

Usage

```
char* asctime_r (const struct tm *timeptr,  
                char *buf)
```

Arguments

- `timeptr`
Pointer to the `tm` structure.
- `buf`
User supplied buffer to be filled with the converted time.

Return Value

- `char*`
Pointer to the converted time if successful.
- `NULL`
Not successful.

Example

```
#include "services/time.h"  
  
char *convtime;  
char buf[40];  
struct tm time_now;  
  
time_now.tm_wday = 03;  
time_now.tm_mon  = 07;  
time_now.tm_mday = 21;  
time_now.tm_hour = 05;  
time_now.tm_min  = 59;  
time_now.tm_sec  = 23;  
time_now.tm_year = 73;  
  
/* Convert the time into asctime */  
convtime = asctime_r(&time_now, (char*)buf);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [clock](#)
[ctime](#) [difftime](#)
[gmtime](#) [localtime](#)

[mktime](#)

[time](#)

[<time.h>](#)

[strftime](#)

[utime](#)

[Nucleus POSIX Run Time Library APIs](#)

asin

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the principal value of the arc sines of their argument x. The value of x should be in the [1,1] range.

Usage

```
double asin (double x)
```

Arguments

- x
The argument converted to arc sine.

Return Value

- double
Arc sine of x will be returned in the $[-\pi/2, \pi/2]$ radians range upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 1;
...
result = asin(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <math.h>

[sin](#)

[Nucleus POSIX Run Time Library APIs](#)

atan

This function computes the principal value of the arc tangents of their argument x.

Usage

```
double atan (double x)
```

Arguments

- x
The argument converted to arc tangent.

Return Value

- double
Arc tangent of x will be returned in the $[-\pi/2, \pi/2]$ radians range upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 1;
...
result = atan(target);
...
```

Related Topics

[atan2](#)

[<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[tan](#)

atan2

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the principal value of the arc tangent of y/x , using the sines of both arguments to determine the quadrant of the return value.

Usage

```
double atan2 (double y,  
             double x)
```

Arguments

- **y**
Y coordinate.
- **x**
X coordinate

Return Value

- **double**
Arc tangent of x will be returned in the $[-\pi, \pi]$ radians range upon successful completion.

Example

```
#include "services/math.h"  
  
static double x, y;  
static double result;  
...  
x = 1;  
y = 1;  
...  
result = atan2(x, y);  
...
```

Related Topics

[acos](#)

[atan](#)

[hypot](#)

[sqrt](#)

[asin](#)

Base Definitions volume of IEEE Std 1003.1-2001

[<math.h>](#)

[tan](#)

[Nucleus POSIX Run Time Library APIs](#)

atof

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by str to double. This is equivalent to:

```
strtod(str, (char **) NULL);
```

Usage

```
double atof (const char *str)
```

Arguments

- str
Pointer to the string converted into float.

Return Value

- double
Return the converted value into float.

Example

```
#include "services/string.h"
#include "services/stdlib.h"

static char string[6];
static double result;
...
strcpy(string, "abcdef");
...
result = atof(string);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[strtod](#)

[Nucleus POSIX Run Time Library APIs](#)

atoi

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by str to integer. The call atoi(str) is equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```

Usage

```
int atoi (const char *str)
```

Arguments

- str
Pointer to the string converted to integer.

Return Value

- int
Returns the converted value to integer.

Example

```
#include "services/string.h"
#include "services/stdlib.h"

static char string[6];
static int result;
...
strcpy(string, "abcdef");
...
result = atoi(string);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[strtol](#)

[Nucleus POSIX Run Time Library APIs](#)

atoi

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by str to long.

The call atoi(str) is equivalent to:

```
strtol(str, (char **)NULL, 10)
```

Usage

```
long atoi (const char *str)
```

Arguments

- str
Pointer to the string converted to long.

Return Value

- long
Return the converted value to long.

Example

```
#include "services/stdlib.h"
#include "services/string.h"

static char string[6];
static long result;
...
strcpy(string, "abcdef");
...
result = atoi(string);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[strtol](#)

[Nucleus POSIX Run Time Library APIs](#)

atoll

This function converts a string to a long integer.

Usage

```
long long atoll(const char *nptr)
```

Arguments

- `nptr`
Pointer to the number string.

Return Values

- `long long`
The converted value if the value can be represented.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

bcmp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function compares the first n bytes of the area pointed to by s1 with the area pointed to by s2.

Usage

```
int bcmp (const void *s1,
          const void *s2,
          size_t      n)
```

Arguments

- s1
Pointer to the first string.
- s2
Pointer to the second string.
- n
Number of bytes to compare.

Return Value

- 0
If s1 and s2 are identical or if the value of n is zero.
- Non zero
If s1 and s2 are not identical.

Example

```
#include "services/string.h"
#include "services/strings.h"

static char string1[6], string2[6];
static int flag;

...
strcpy (string1, "abAde");
strcpy (string2, "abAde");
...

flag = bcmp(string1, string2, 4);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[memcmp](#)

<strings.h>

[Nucleus POSIX Run Time Library APIs](#)

bcopy

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copies n bytes from the area pointed to by s1 to the area pointed to by s2.

Usage

```
void bcopy (const void *s1,  
            void        *s2,  
            size_t      n)
```

Arguments

- s1
Pointer to the first string.
- s2
Pointer to the second string.
- n
Number of bytes to copy.

Example

```
#include "services/string.h"  
#include "services/strings.h"  
  
static char string1[6], string2[6];  
...  
strcpy (string1, "abAde");  
strcpy (string2, "-----");  
...  
  
bcopy(string1, string2, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [memmove](#)

<strings.h>

[Nucleus POSIX Run Time Library APIs](#)

bsearch

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function searches an array of `nel` objects, the initial element of which is pointed to by `base`, for an element that matches the object pointed to by `key`. The size of each element in the array is specified by `width`.

Usage

```
void *bsearch (const void *key,
               const void *base,
               size_t     nel,
               size_t     width,
               int         (*compare) (const void*, const void*))
```

Arguments

- `key`
Pointer to the key object.
- `base`
Pointer to the element base.
- `nel`
Number of elements.
- `width`
Size of each element.
- `compare`
Pointer to the comparison function.

Return Value

- `void*`
Returns the pointer to a matching member of the array.
- `NULL`
No match is found.

Example

```
#include "services/stdio.h"
#include "services/stdlib.h"
#include "services/string.h"

#define TABSIZE 1000

struct node { /* These are stored in the table. */
    char *string;
    int length;
};
```



```
struct node table[TABSIZE]; /* Table to be searched. */
...
{
    struct node *node_ptr, node;

    /* Routine to compare 2 nodes. */
    int node_compare(const void *, const void *);
    char str_space[20]; /* Space to read string into. */
    ...
    node.string = str_space;
    while (scanf("%s", node.string) != EOF)
    {
        node_ptr = (struct node *)bsearch((void *)(&node),
                                           (void *)table, TABSIZE,
                                           sizeof(struct node), node_compare);
        if (node_ptr != NULL)
        {
            (void)printf("string = %20s, length = %d\n",
                        node_ptr->string, node_ptr->length);
        }
        else
        {
            (void)printf("not found: %s\n", node.string);
        }
    }
}

/* This routine compares two nodes based on an alphabetical ordering of
   the string field.*/

int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
                  ((const struct node *)node2)->string);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[Nucleus POSIX Run Time Library APIs](#)

bzero

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function places n zero-valued bytes in the area pointed to by s.

Usage

```
void bzero (void    *s,  
            size_t n)
```

Arguments

- s
Pointer to the string.
- n
Number of bytes to be set to zero valued.

Example

```
#include "services/string.h"  
#include "services/strings.h"  
  
static char string1[6];  
...  
strcpy (string1, "abAde");  
...  
  
bzero(string1, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [memset](#)

<strings.h>

[Nucleus POSIX Run Time Library APIs](#)

calloc

Allowed from: POSIX Thread and signal handler

This function allocates unused space for an array of `nelem` elements each of whose size in bytes is `elsize`. Setting all bits to 0 initializes the space.

Usage

```
void *calloc (size_t nelem,  
             size_t elsize)
```

Arguments

- `nelem`
Number of elements.
- `elsize`
Element size.

Return Value

- `void*`
Pointer to the allocated space.
- `NULL`
An error is encountered or either `nelem` or `elsize` is 0. `errno` is set to the corresponding error code:

 ENOMEM
 Insufficient storage space is available.

Example

```
#include "services/stdlib.h"  
  
static int *target;  
...  
target = (int *)calloc(5, sizeof(int));  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [free](#)

[malloc](#)

[realloc](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

ceil

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the smallest integral value not less than x .

Usage

```
double ceil (double x)
```

Arguments

- x
The argument provided.

Return Value

- double
Smallest integral value no less than x upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 5.6;
...
result = ceil(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<math.h>

[floor](#)

[Nucleus POSIX Run Time Library APIs](#)

clearerr

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function clears the end-of-file and error indicators for the stream to which the stream points.

Usage

```
void clearerr (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Example

```
#include "services/stdio.h"

clearerr(stdin);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

clock

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function returns the implementation's best approximation to the processor time used by the process since the beginning of an implementation-defined era related only to the process invocation.

Usage

```
clock_t clock (void)
```

Return Value

- -1
Processor time used is not available.

Example

```
#include "services/time.h"

clock_t clk;

/* Find the processor time used by the process */
clk = clock();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	asctime
ctime	difftime
gmtime	localtime
mktime	strftime
time	utime
<time.h>	Nucleus POSIX Run Time Library APIs

clock_nanosleep

This function provides high resolution sleep with specifiable clock.

Usage

```
int clock_nanosleep(clockid_t      clock_id,  
                    int            flags,  
                    const struct timespec *rqtp,  
                    struct timespec *rmtp)
```

Arguments

- **clock_id**
The clock used to measure time.
- **flags**
Flags value that effect the operation.
- **rqtp**
Specifies the amount of time to sleep.
- **rmtp**
The amount of time remaining in the interval if a relative clock_nanosleep() function call is interrupted.

Return Values

- **0**
The requested time has elapsed.
- **EINTR**
The clock_nanosleep() function was interrupted by a signal.
- **EINVAL**
The rqtp argument specified a nanosecond value less than zero or greater than or equal to 1000 million; or the **TIMER_ABSTIME** flag was specified in flags and the rqtp argument is outside the range for the clock specified by clock_id; or the clock_id argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.
- **ENOTSUP**
The clock_id argument specifies a clock for which clock_nanosleep() is not supported, such as a CPU-time clock.

Description

For the relative clock_nanosleep() function, if the rmtp argument is non-NULL, the timespec structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). If the rmtp argument is NULL, the remaining time is not returned.

The absolute `clock_nanosleep()` function has no effect on the structure referenced by `rmtp`.

Example

```
#include "services/unistd.h"
#include "services/pthread.h"
#include "services/time.h"
#include "services/signal.h"
#include "services/stdio.h"

/* Global Variables */
static pthread_t Test_Psx_Thread2;
static int Posix_Error = 1;
static int Psx_Clock_Nano_Counter = 0;

/* Testing clock_nanosleep. Waiting for the absolute time to end. */
static void* Test_Psx_Thread2_Entry(void *argc)
{
    int status;
    struct timespec rqtp;
    struct timespec rmtp;

    rqtp.tv_sec = 3;
    rqtp.tv_nsec = 1000000;

    status = clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &rqtp, &rmtp);
    if (status == POSIX_ERROR)
    {
        printf("error error count: %i, line: %i \n", Posix_Error++, __LINE__);
    }
    else
    {
        Psx_Clock_Nano_Counter++;
    }

    if (Psx_Clock_Nano_Counter == 1)
    {
        printf("error error count: %i, line: %i \n", Posix_Error++, __LINE__);
    }

    pthread_exit(NULL);

    return 0;
}
```

Related Topics

clock_getres	nanosleep
pthread_cond_timedwait	sleep
the Base Definitions volume of IEEE Std 1003.1-2001	<time.h>

[Nucleus POSIX Run Time Library APIs](#)

confstr

This function returns configuration-defined string values. Its use and purpose are similar to [sysconf](#), but it is used where string values rather than numeric values are returned.

Usage

```
#include "services\unistd.h"

...
size_t confstr(int    name,
               char    *buf,
               size_t  len)
```

Arguments

- **name**
The name argument represents the system variable to be queried. Only `_CS_PATH` is supported.
- **buf**
The value to be copied into the len-byte buffer point to by buf.
- **len**
Length of the application buffer.

Return Values

- **Size_t**
If name has a configuration-defined value, `confstr()` returns the size of buffer that would be needed to hold the entire configuration-defined value including the terminating null. If this return value is greater than len, the string returned in buf is truncated.
- **0**
If name is invalid, this function returns zero and sets `errno` to the following error code to indicate the error:

 EINVAL

 The value of the name argument is invalid.

If name does not have a configuration-defined value, this function returns zero and leaves `errno` unchanged.

Description

If len is not zero, and if name has a configuration-defined value, `confstr()` copies that value into the len-byte buffer pointed to by buf. If the string to be returned is longer than len bytes, including the terminating null, then `confstr()` truncates the string to len-1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by `confstr()` with len.

If len is zero and buf is a null pointer, then confstr() still returns the integer value, but it does not return a string. If len is zero, but buf is not a null pointer, the result is unspecified.

The Nucleus POSIX product must be successfully initialized before calling this function.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

cos

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the cosine of their argument x, measured in radians.

Usage

```
double cos (double x)
```

Arguments

- x
The x argument.

Return Value

- double
Returns the cosine of x.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 3.14159;
...
result = cos(target);
...
```

Related Topics

[acos](#)

[sin](#)

[<math.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[tan](#)

[Nucleus POSIX Run Time Library APIs](#)

cosh

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the hyperbolic cosine of their argument x.

Usage

```
double cosh (double x)
```

Arguments

- x
The argument provided.

Return Value

- double
Hyperbolic cosine of x will be returned upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 3.14159;
...
result = cosh(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sinh](#)
[tanh](#) [<math.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

ctime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the time pointed to by clock, representing time in seconds since the epoch, to local time in the form of a string. This function is non-reentrant.

Usage

```
char* ctime (const time_t* clock)
```

Arguments

- clock
Pointer to the clock.

Return Value

- char*
Pointer to the converted time if successful.
- NULL
Not successful.

Example

```
#include "services/time.h"

time_t clock;
char    *cnv_time;

/* Set the clock */
clock = 12345;

/* Find the converted time associated with the clock */
cnv_time = ctime(&clock);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	asctime
clock	difftime
gmtime	localtime
mktime	strftime
time	utime
<time.h>	Nucleus POSIX Run Time Library APIs

ctime_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the time pointed to by clock, representing time in seconds since the epoch, to local time in the form of a string. This is the re-entrant version of [ctime](#).

Usage

```
char *ctime_r (const time_t *clock,  
               char          *buf)
```

Arguments

- clock
Pointer to the clock.
- buf
User supplied buffer to be filled with the converted time.

Return Value

- char*
Pointer to the converted time if successful.
- NULL
Not successful.

Example

```
#include "services/time.h"  
  
time_t clock;  
char    buf[40];  
  
/* Set the clock */  
clock = 12345;  
  
/* Find the converted time associated with the clock */  
cnv_time = ctime_r(&clock, (char*)buf);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	asctime
clock	difftime
gmtime	localtime
mktime	strftime
time	utime
<time.h>	Nucleus POSIX Run Time Library APIs

difftime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the difference between two calendar times.

Usage

```
double difftime (time_t time1,  
                 time_t time0)
```

Arguments

- `time1`
First value of time.
- `time0`
Second value of time.

Return Value

- `double`
The difference expressed in seconds as a type double.

Example

```
#include "services/time.h"  
  
time_t time_1;  
time_t time_2;  
double diff_time;  
  
time_2 = 4567;  
time_1 = 1234;  
  
/* Find the difference between these two times in second */  
diff_time = difftime(time_2,time_1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	asctime
clock	ctime
gmtime	localtime
mktime	strftime
time	utime
<time.h>	Nucleus POSIX Run Time Library APIs

div

This function computes the quotient and remainder of an integer division.

Usage

```
div_t div(int numer,  
          int denom)
```

Arguments

- **numer**
The numerator.
- **denom**
The denominator.

Return Values

- **div_t**
Division type structure.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

exp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the base-e exponential of x.

Usage

```
double exp (double x)
```

Arguments

- x
The argument provided.

Return Value

- double
Exponential value of x will be returned upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 0;
...
result = exp(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [log](#)

[<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

fabs

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the absolute value of their argument x , $|x|$.

Usage

```
double fabs (double x)
```

Arguments

- x
The argument provided.

Return Value

- double
Absolute value of x will be returned upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = -5.0
...
result = fabs(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 `<math.h>`

[Nucleus POSIX Run Time Library APIs](#)

fclose

Allowed from: POSIX Thread and signal handler

This function flushes the specified stream and closes the associated file.

Usage

```
int fclose (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- 0
Function call is successful.
- EOF
If an error is encountered, `errno` is set to the corresponding error code:

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

There was no free space remaining on the device containing the file.

Example

```
#include "services/stdio.h"

FILE *stream;

if (stream = fopen("data", "r") == NULL)
    printf("The file data was not opened \n");

if (fclose(stream))
    printf("The file data was not closed \n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

[fopen](#)

Interaction of File Descriptors and Standard I/O Streams

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

fdopen

Allowed from: POSIX Thread and signal handler

This function associates a stream with a file descriptor.

Usage

```
FILE *fdopen (int      fildes,  
              const char *mode)
```

Arguments

- fildes

Target file stream.

- mode

Opening mode. This can be set to one of the following values:

r or rb

Open a file for reading.

w or wb

A file for writing.

a or ab

A file for writing at the end of the file.

r+ or rb+ or r + b

A file for update (Reading and Writing).

w+ or wb+ or w + b

Open a file for update (Reading and Writing).

a+ or ab+ or a + b

Open a file for update (Reading and Writing) at end of file.

Return Value

- stream

Function call is successful.

- NULL

If an error is encountered, errno is set to the corresponding error code:

EINVAL

The mode argument is not a valid mode.

EMFILE

{FOPEN_MAX} streams are currently open in the calling process.

ENFILE

{STREAM_MAX} streams are currently open in the calling process.

ENOMEM

Insufficient space to allocate a buffer.

Example

```
#include "services/fcntl.h"
#include "services/stdio.h"

FILE *stream;
int fd;

fd = open("file1", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR);

if ( (stream = fdopen(fd, "wb") ) == NULL)
    printf("fdopen() fails \n");
else
    printf("Call to fdopen() is successful \n")
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fopen](#)

[open](#)

[<stdio.h>](#)

Interaction of File Descriptors and Standard I/O Streams

[socket](#)

[Nucleus POSIX Run Time Library APIs](#)

feclearexcept

This function clears the floating-point exception (non-functional stub only).

Usage

```
int feclearexcept (int excepts)
```

Arguments

- `excepts`
The exceptions to clear.

Return Values

- `int`
A zero value is returned if the exceptions were successfully cleared. A non-zero value is returned otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fegetenv

This function gets the current floating-point environment (non-functional stub only).

Usage

```
int fegetenv(fenv_t *envp)
```

Arguments

- `envp`
Pointer to object that will contain the current floating-point environment.

Return Values

- `int`
A zero value is returned if the operation is successful and a non-zero value otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fegetexceptflag

This function gets the floating-point status flags (non-functional stub only).

Usage

```
int fegetexceptflag(fexcept_t *flagp,  
                   int         excepts)
```

Arguments

- **flagp**
Pointer to object that will contain the floating-point flags.
- **excepts**
Indicates the exceptions to be stored in the object pointed to by flagp.

Return Values

- **int**
A zero value is returned if the operation is successful and a non-zero value otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fegetround

This function retrieves the current rounding direction (non-functional stub only).

Usage

```
int fegetround(void)
```

Return Values

- int

This is the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro. It can also be the value of the rounding direction macro if the current rounding direction is not determinable.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

feholdexcept

This function saves the current floating-point environment (non-functional stub only).

Usage

```
#include <fenv.h>
...
int feholdexcept(fenv_t *envp)
```

Arguments

- `envp`
Pointer to floating-point environment structure to which to save.

Return Values

- `int`
A zero value is returned if non-stop floating-point exception handling was successfully installed. A non-zero value is returned otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

feof

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests the end-of-file indicator for the stream pointed to by stream.

Usage

```
int feof (FILE *stream)
```

Arguments

- stream
Target file stream.

Return Value

- Non-zero
If and only if the end-of-file indicator is set for stream.

Example

```
#include "services/stdio.h"

FILE *stream;
int eof_flag;

stream = fopen("stream1", "r");

/* Test the EOF flag for the stream */
eof_flag = feof(stream);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [clearerr](#)

[ferror](#) [fopen](#)

[<stdio.h>](#) [Nucleus POSIX Run Time Library APIs](#)

feraiseexcept

This function raises the floating-point exception (non-functional stub only).

Usage

```
int ferroraiseexcept(int excepts)
```

Arguments

- `excepts`
The exceptions to raise.

Return Values

- 0
A zero value is returned if the exceptions were successfully raised.
- Non-zero value
A non-zero value is returned if there the exceptions were not successfully raised.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

ferror

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests the error indicator for the stream pointed to by stream.

Usage

```
int ferror (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- Non-zero
If and only if the error indicator is set for stream.

Example

```
#include "services/stdio.h"

FILE *stream;
int  error_flag;

stream = fopen("stream1", "r");

/* Test the EOF flag for the stream */
error_flag = ferror(stream);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [clearerr](#)

[feof](#) [fopen](#)

[<stdio.h>](#) [Nucleus POSIX Run Time Library APIs](#)

fesetenv

This function sets the current floating-point environment (non-functional stub only).

Usage

```
int fesetenv(const fenv_t *envp)
```

Arguments

- `envp`
Pointer to object that contains the floating-point environment to be set.

Return Values

- `int`
A zero value is returned if the operation is successful and a non-zero value otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fesetexceptflag

This function sets the floating-point status flags (non-functional stub only).

Usage

```
int fesetexceptflag(const fexcept_t *flagp,  
                    int             excepts)
```

Arguments

- **flagp**
Pointer to object that will contain the floating-point flags to be set.
- **excepts**
Indicates the exceptions to be set in the system based on the values in the object pointed to by flagp.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fesetround

This function sets the current rounding direction (non-functional stub only).

Usage

```
int fesetround(int round)
```

Arguments

- **round**

The rounding direction to be set. This must match the value of a rounding direction macro or nothing happens.

Return Values

- **int**

Zero if the operation is successful and non-zero otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fetestexcept

This function tests the exceptions in the floating-point system (non-functional stub only).

Usage

```
int fetestexcept(int excepts)
```

Arguments

- `excepts`
The exceptions to be tested.

Return Values

- `int`
Value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in `excepts`.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

feupdateenv

This function updates the current floating-point environment (non-functional stub only).

Usage

```
int feupdateenv(const fenv_t *envp)
```

Arguments

- `envp`
Pointer to an object that will contain the floating-point environment to be used.

Return Values

- `int`
A zero value is returned if the operation is successful and a non-zero value otherwise.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

fflush

Allowed from: POSIX Thread and signal handler

This function causes any unwritten data for that stream to be written to the file.

Usage

```
int fflush (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- 0
Function call is successful.
- EOF
If error is encountered and errno is set to the corresponding error code:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

There was no free space remaining on the device containing the file.

EINVAL

The stream argument is not a valid stream.

EACCES

Permission denied.

Example

```
#include "services/stdio.h"

int status;

/* Flush the stdout stream */
status = fflush(stdout);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

ffs

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function finds the first bit set (beginning with the least significant bit) in *i*, and returns the index of that bit. Bits are numbered starting at one (the least significant bit).

Usage

```
int ffs (int i)
```

Arguments

- *i*
Argument to be searched for the first set bit.

Return Value

- *int*
Index of the first bit set.
- 0
If *i* is zero.

Example

```
#include "services/strings.h"

static int target, indexOfBit;
...
target = 0x00100000;
...
indexOfBit = ffs(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- <strings.h>
2001

[Nucleus POSIX Run Time Library APIs](#)

fgetc

Allowed from: POSIX Thread and signal handler

This function obtains the next byte as an unsigned character converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream.

Usage

```
int fgetc (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- stream
Next byte from the input stream pointed to by stream.
- EOF
If error is encountered and errno is set to the corresponding error code:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;

/* Get a character from stdin */
ch = fgetc(stdin);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [feof](#)

[ferror](#)

[getc](#)

[<stdio.h>](#)

[fopen](#)

[getchar](#)

[Nucleus POSIX Run Time Library APIs](#)

fgetpos

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function stores the current values of the parse state (if any) and file position indicator for the stream pointed to by stream in the object pointed to by pos.

Usage

```
int fgetpos (FILE *stream,
             fpos_t *pos)
```

Arguments

- stream
Pointer to the file stream.
- pos
Pointer to the file position.

Return Value

- 0
Function call successful.
- Non-zero

If error is encountered and errno is set to the corresponding error code:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

There was no free space remaining on the device containing the file.

EINVAL

The stream argument is not a valid stream.

EACCES

Permission denied.

Example

```
#include "services/stdio.h"

FILE *stream;
fpos_t pos;
char buffer[20];

if ((stream = fopen("stream1", "rb") ) == NULL)
    printf("Trouble opening the file \n");
```



```
else
{
    /* Read some data and then check the position */
    fread(buffer,sizeof(char),10,stream);

    if (fgetpos(stream,&pos) != 0)
        perror("fgetpos error \n");
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	fopen
ftell	rewind
ungetc	<stdio.h>
Nucleus POSIX Run Time Library APIs	

fgets

Allowed from: POSIX Thread and signal handler

This function reads bytes from stream into the array pointed to by s, until n-1 bytes are read, or a <newline> is read and transferred to s, or an end-of-file condition is encountered.

Usage

```
char *fgets (char *s,  
             int  n,  
             FILE *stream)
```

Arguments

- s
Buffer to read.
- n
Number of bytes to read.
- stream
Pointer to the file stream.

Return Value

- char*
Pointer to the get string if function call successful.
- NULL

If an error is encountered, errno is set to the corresponding error code:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EINVAL

The stream argument is not a valid stream.

EACCES

Permission denied.

Example

```
#include "services/stdio.h"  
  
FILE *stream;  
char line[100];
```

```
if ( (stream = fopen("stream1", "r")) != NULL)
{
    if (fgets(line, 100, stream) == NULL)
        printf("fgets error \n");
    else
        printf("%s", line);

    fclose(stream);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fread](#)

[<stdio.h>](#)

[fopen](#)

[gets](#)

[Nucleus POSIX Run Time Library APIs](#)

fileno

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function returns the integer file descriptor associated with the stream pointed to by stream.

Usage

```
int fileno (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- int
Integer value of the file descriptor associated with stream if the function call is successful.
- POSIX_ERROR
If the function call fails. It then set corresponding errno to indicate the error:
EBADF
The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

printf("The file handle for stdin is %d \n", fileno(stdin));
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fopen](#)

<stdio.h>

[fdopen](#)

Interaction of File Descriptors and Standard I/O Streams

[Nucleus POSIX Run Time Library APIs](#)

flockfile

Allowed from: POSIX Thread and signal handler

This function acquires a thread for ownership of a (FILE *) object.

Usage

```
void flockfile (FILE *file)
```

Arguments

- file
Pointer to the file stream.

Example

```
#include "services/stdio.h"

FILE *file;
file = fopen("test.txt", "r+b");

/* locks a file, counter incremented */
flockfile(file);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc_unlocked](#)
[putc_unlocked](#) [<stdio.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

floor

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the largest integral value not greater than x.

Usage

```
double floor (double x)
```

Arguments

- x
The argument provided.

Return Value

- double
Largest integral value no greater than x will be returned upon successful completion.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 5.6;
...
result = floor(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [ceil](#)

[<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

fmod

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function returns the floating-point remainder of the division of x by y.

Usage

```
double fmod (double x,  
            double y)
```

Arguments

- x
The x coordinate.
- y
The y coordinate

Return Value

- double
Returns the value $x_i * y$, for some integer i if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

Example

```
#include "services/math.h"  
  
static double x, y;  
static double result;  
...  
x = 9;  
y = 4;  
...  
result = fmod(x, y);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <math.h>

[Nucleus POSIX Run Time Library APIs](#)

fopen

Allowed from: POSIX Thread and signal handler

This function opens the file whose pathname is the string pointed to by filename, and associates a stream with it.

Usage

```
FILE *fopen (const char *filename,  
             const char *mode)
```

Arguments

- filename

Name of the file to open.

- mode

Mode in which the file is to be opened. This can be set to one of the following values:

r or rb

Open a file for reading.

w or wb

Open a file for writing.

a or ab

Open a file for writing at the end of the file.

r+ or rb+ or r + b

Open a file for update (Reading and Writing).

w+ or wb+ or w + b

Open a file for update (Reading and Writing).

a+ or ab+ or a + b

Open a file for update (Reading and Writing) at end of file.

Return Value

- FILE *

On successful completion, a pointer to the object controlling the stream is returned.

- NULL

A null pointer is returned when the function call fails. It then sets errno to the corresponding code to indicate the error.

EACCES

Permission denied.

EMFILE

{OPEN_MAX} file descriptors are currently open in the calling process.

ENAMETOOLONG

The length of the filename argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

ENOENT

A component of filename does not name an existing file or filename is an empty string.

ENOSPC

The directory or file system that would contain the new file cannot be expanded, the file does not exist, and the file was to be created.

EINVAL

The value of the mode argument is not valid.

ENAMETOOLONG

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

ENOMEM

Insufficient storage space is available.

EEXIST

The named file exists.

EIO

An I/O error occurred while accessing the file system.

Example

```
#include "services/stdio.h"

FILE* stream;

if ( (stream = fopen("data","r") ) == NULL)
    printf("The file data was not opened \n");
else
    printf("The file data was opened \n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fclose](#)

[fdopen](#)

[freopen](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

fpathconf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function determines the current value of a configurable limit or option (variable) associated with a file or directory.

Usage

```
long fpathconf (int fildes,  
               int name)
```

Arguments

- **fildes**
Open file descriptor.
- **name**
Variable to be queried relative to that file or directory.

Return Value

- **POSIX_ERROR**
Error encountered. In this case `errno` is set to the corresponding error code:

EINVAL
The value of `name` is invalid.
- **long**
Configurable limit associated with the file or directory on success.

Example

```
#include "services/unistd.h"  
  
int fd;  
long value;  
  
/* Find the Maximum Name Length limit associated with the  
   file "/usr/file1.txt" */  
fd = open("/usr/file1.txt", O_RDONLY);  
  
...  
if (fd != -1)  
{  
    value = fpathconf(fd, NAME_MAX);  
}  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<limits.h>](#)

Shell and Utilities volume of IEEE Std 1003.1-2001 [sysconf](#)

<unistd.h>

Nucleus POSIX Run Time Library APIs

fprintf

Allowed from: POSIX Thread and signal handler

This function places output on the named output stream.

Usage

```
int fprintf (FILE          *stream,  
             const char *format,  
             ...)
```

Arguments

- stream
Pointer to the file stream.
- format
Output formatting string.

Supported Flag Characters

- -
The result of the conversion will be left justified within the field.
- 0
For d, i, o, u, x, X, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width.

Supported Length Modifiers

- l
Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- d,i
The int argument is converted to a signed decimal in the style “[]dddd”.
- o
The unsigned argument is converted to unsigned octal format in the style “dddd”.
- u
The unsigned argument is converted to unsigned decimal format in the style “dddd”.
- x
The unsigned argument is converted to unsigned hexadecimal format in the style “dddd”.

- **X**
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.
- **f,F**
The double argument is converted to decimal notation in the style “[ddd.ddd]”.
- **e,E**
The double argument is converted in the style “[d.ddde±dd]”.
- **g,G**
The double argument is converted in the style f or e (or in the style F or E in the case of a G conversion specifier).
- **c**
The int argument is converted to an unsigned char, and the resulting byte will be written.
- **s**
The argument is a pointer to an array of char. Bytes from the array will be written up to (but not including) any terminating null byte.
- **%**
Print a ‘%’ character; no argument is converted. The complete conversion specification is%%.

Return Value

- **int**
Number of bytes on successful completion.
- **Negative value**
An error is encountered. In this case errno is set to the corresponding error code:
 - EINVAL**
There are insufficient arguments.
 - ENOMEM**
Insufficient storage space is available.

Example

```
#include "services/stdio.h"

FILE *stream;
chars[] = "This is a string";
char c = '\n';

stream = fopen("stream1", "w");
fprintf(stream, "%s %c", s, c);
fclose(stream);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fputc](#)

[fscanf](#)

[<stdio.h>](#)

[<wchar.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

fputc

Allowed from: POSIX Thread and signal handler

This function writes the byte specified by *c* (converted to an unsigned char) to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately.

Usage

```
int fputc (int c,
           FILE *stream)
```

Arguments

- *c*
Specifies the byte to be written.
- *stream*
Pointer to the file stream.

Return Value

- *int*
On successful completion, returns the value it has written.
- EOF

If the call fails it returns EOF, and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
char strptr[] = "This is a test of putc()!! \n";
char *p;
int ch;
```

```
ch = 0;
stream = stdout;

for(p = buffer; (ch != EOF) && (*p != '\\0'); p++)
    ch = fputc(*p, stream);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fopen](#)

[puts](#)

[<stdio.h>](#)

[ferror](#)

[putc](#)

[setbuf](#)

[Nucleus POSIX Run Time Library APIs](#)

fputs

Allowed from: POSIX Thread and signal handler

This function writes the null-terminated string pointed to by *s* to the stream pointed to by *stream*.

Usage

```
int fputs (const char *s,  
           FILE      *stream)
```

Arguments

- *s*
String to be put on the stream.
- *stream*
Pointer to the file stream.

Return Value

- *int*
On successful completion, returns a non-negative number.
- EOF
If the call fails, it returns EOF and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"  
  
fputs("Hello world from fputs. \n",stdout);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fopen](#)

[putc](#)

[<stdio.h>](#)

[puts](#)

[Nucleus POSIX Run Time Library APIs](#)

fread

Allowed from: POSIX Thread and signal handler

The function reads into the array pointed to by ptr up to nitems elements whose size is specified by size in bytes, from the stream pointed to by stream.

Usage

```
size_t fread (void    *ptr,  
              size_t  size,  
              size_t  nitems,  
              FILE    *stream)
```

Arguments

- ptr
String to be put on the stream.
- size
Size of elements to be read in bytes.
- nitems
Number of items to be read.
- stream
Pointer to the file stream.

Return Value

- size_t
On successful completion, returns the number of bytes read.
- Number less than nitems
If the call fails, it returns a number less than nitems and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;

char list[30];
int numread;

if ( (stream = fopen("stream1","r + t")) != NULL)
{
    /* Attempt to read in 25 characters */
    numread = fread(list,sizeof(char),25,stream);

    fclose(stream);
}
else
    printf("This file could not be opened \n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	feof
ferror	fgetc
fopen	getc
gets	scanf
<stdio.h>	Nucleus POSIX Run Time Library APIs

free

Allowed from: POSIX Thread and signal handler

This function deallocates the space pointed to by ptr. This makes the space available for further allocation.

Usage

```
void free (void *ptr)
```

Arguments

- ptr
Pointer to the memory object freed.

Description

The contents of the object remain unchanged up to the lesser of the new and old sizes. If the new size of the memory object requires movement of the object, the space for the previous instantiation of the object is freed. If the new size is larger, the contents of the newly allocated portion of the object are unspecified. If size is 0 and ptr is not a null pointer, the object pointed to is freed. If the space cannot be allocated, the object remains unchanged.

Example

```
#include "services/stdlib.h"

static int *target;
...
target = (int *)malloc(sizeof(int));
...
free(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[malloc](#)

[<stdlib.h>](#)

[calloc](#)

[realloc](#)

[Nucleus POSIX Run Time Library APIs](#)

frexp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function breaks a floating-point number `num` into a normalized fraction and an integral power of 2. The integer exponent is stored in the `int` object pointed to by `exp`.

Usage

```
double frexp (double num,  
              int  *exp)
```

Arguments

- `num`
The floating point number to be broken.
- `exp`
Pointer to an integer where integer exponent is stored.

Return Value

- `double`
Returns the value `x`, such that `x` has a magnitude in the interval $[1/2, 1)$ or 0, and `num` equals `x` times 2 raised to the power `*exp`.

Example

```
#include "services/math.h"  
  
static double target, result;  
static int exp;  
...  
target = 5.679;  
...  
frexp(target, &exp);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [ldexp](#)
[modf](#) [<math.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

freopen

Allowed from: POSIX Thread and signal handler

This function opens the file whose pathname is the string pointed to by the filename and associate the stream pointed to by the stream with it.

Usage

```
FILE *freopen(const char *restrict filename,
              const char *restrict mode,
              FILE *restrict stream)
```

Arguments

- filename
Pointer to the filename to be opened.
- mode
Mode in which the file is to be opened. This can be set to one of the following values:
 - r or rb
Open a file for reading.
 - w or wb
Open a file for writing.
 - a or ab
Open a file for writing at the end of the file.
 - r+ or rb+ or r + b
Open a file for update (Reading and Writing).
 - w+ or wb+ or w + b
Open a file for update (Reading and Writing).
 - a+ or ab+ or a + b
Open a file for update (Reading and Writing) at end of file.
- stream
Pointer to the existing stream.

Return Value

- filename
Upon successful completion, freopen() returns the value of the stream.
- NULL
If the call fails, it returns a null pointer EOF and sets errno to the corresponding error:
EACCES

Permission denied.

EBADF

The file descriptor-underlying stream is not valid.

EINVAL

The stream argument is not a valid stream.

EIO

I/O error.

ENOSPC

There was no free space remaining on the device containing the file.

EMFILE

{OPEN_MAX} file descriptors are currently open in the calling process.

ENAMETOOLONG

The length of the filename argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

ENOENT

A component of filename does not name an existing file or filename is an empty string.

ENOSPC

The directory or file system that would contain the new file cannot be expanded, the file does not exist, and the file was to be created.

EINVAL

The value of the mode argument is not valid.

ENAMETOOLONG

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

ENOMEM

Insufficient storage space is available.

EEXIST

The named file exists.

EIO

An I/O error occurred while accessing the file system.

EINTR

A signal was caught during freopen().

ENFILE

The maximum allowable number of files is currently open in the system.

EOverflow

The named file is a regular file and the size of the file cannot be represented correctly in an object of type `off_t`.

Example

```
#include "services/stdio.h"
FILE *fp1;
FILE *fp;
if ((fp1 = fopen("./frpn-t.7", "a+")) == (FILE *) NULL)
{
    perror("fopen() failed");
}

if ((fp = freopen("./frpn-t.7a", "r+", fp1)) == (FILE *) NULL)
{
    perror("freopen() failed");
    fclose(fp1);
    unlink("./frpn-t.7");
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fclose](#)

[fdopen](#)

[fopen](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

fscanf

Allowed from: POSIX Thread and signal handler

This function reads from the named input stream.

Usage

```
int fscanf (FILE      *stream,  
            const char *format,  
            ...)
```

Arguments

- stream
Pointer to the input stream.
- format
Formatting parameters.

Supported Length Modifiers

- l
Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or unsigned long argument.
- h
Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short.

Supported Conversion Specifiers

- d
Matches an optionally signed decimal integer.
- i
Matches an optionally signed integer.
- o
Matches an optionally signed octal integer.
- u
Matches an optionally signed decimal integer.
- x
Matches an optionally signed hexadecimal integer.
- a,e,f,g
Matches an optionally signed floating-point number.

- **s**
Matches a sequence of bytes that are not white-space characters.
- **[**
Matches a non-empty sequence of bytes from a set of expected bytes (the scanset).
- **c**
Matches a sequence of bytes of the number specified by the field width.
- **%**
Matches a single ‘%’ character; no conversion or assignment occurs. The complete conversion specification is%%.

Return Value

- **int**
Number of successfully matched and assigned input items.
- **EOF**
If error encountered.

Example

```
#include "services/stdio.h"

fscanf(stdin, "%s");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getc
printf	strtod
strtol	strtoul
<stdio.h>	<wchar.h>

[Nucleus POSIX Run Time Library APIs](#)

fseek

Allowed from: POSIX Thread and signal handler

This function sets the file-position indicator for the stream pointed to by stream. The new position, measured in bytes from the beginning of the file is obtained by adding offset to the position specified by whence.

Note



This functionality is not supported for the I/O devices.

Usage

```
int fseek (FILE *stream,
           long offset,
           int  whence)
```

Arguments

- stream
Pointer to the file stream.

- offset
Offset from the start.

- whence
Determines from where to seek. This can be set to one of the following values:

SEEK_SET

The file offset is set to offset bytes.

SEEK_CUR

The file offset is set to offset bytes.

SEEK_END

The file offset is set to the size of file plus the offset. Seeking is not allowed beyond the end of the file.

Return Value

- POSIX_SUCCESS
Function call successful.

- POSIX_ERROR
If the call fails, it returns -1 and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
char line[81];
int result;

stream = fopen("stream1", "w+");

if (stream == NULL)
    printf("The file stream1 was not opened \n");

result = fseek(stream, 23L, SEEK_SET);

if (result)
    perror("fseek() failed");

fclose(stream);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fopen](#)

[fsetpos](#)

[ftell](#)

[lseek](#)

[rewind](#)

[<stdio.h>](#)

[ungetc](#)

[write](#)

[Nucleus POSIX Run Time Library APIs](#)

fsetpos

Allowed from: POSIX Thread and signal handler

This function stores the current values of the parse state (if any) and file position indicator for the stream pointed to by stream in the object pointed to by pos.

Note



This functionality is not supported for the streams corresponding to I/O devices.

Usage

```
int fsetpos (FILE          *stream,
            const fpos_t *pos)
```

Arguments

- stream
Pointer to the file stream.
- pos
Position of the file pointer.

Return Value

- 0
Function call successful.
- Non-zero

If the call fails, it returns a non-zero value and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
fpos_t pos;
```

```
if (( stream = fopen("stream1","rb")) == NULL)
    printf("Trouble Opening File \n");
else
{
    pos = 140;

    /* Set the stream position */
    if (fsetpos(stream,&pos) != 0)
        perror("fsetpos error");

    /* Close the stream */
    fclose(stream);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	fopen
ftell	lseek
rewind	ungetc
write	<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

ftell

Allowed from: POSIX Thread and signal handler

This functions gives the current value of the file pointer.

Usage

```
long ftell (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- long
Current position of the file if the call is successful.
- POSIX_ERROR

If the call fails, it returns a non-zero value and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
long position;
char list[100];

if ( (stream = fopen("stream1", "rb") ) != NULL)
{
    /* Move the Pointer by reading data */
    fread(list, sizeof(char), 100, stream);

    /* Get the position after read */
    position = ftell(stream);
}
```



```
    fclose(stream);  
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fgetpos](#)

[fopen](#)

[fseek](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

ftrylockfile

Allowed from: POSIX Thread and signal handler

This function acquires a thread for ownership of a (FILE *) object if the object is available. ftrylockfile() is a non-blocking version of [flockfile](#).

Usage

```
int ftrylockfile (FILE *file)
```

Arguments

- file
Pointer to the file stream.

Return Value

- POSIX_SUCCESS
Successful operation.
- Non-zero
non-zero to indicate that the lock cannot be acquired.

Example

```
#include "services/stdio.h"

FILE *file;
file = fopen("test.txt", "r+b");

/* locks a file, counter incremented */
if (ftrylockfile(file) != 0)
    printf("ftrylockfile failed \n");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc_unlocked](#)

[putc_unlocked](#) [<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

funlockfile

Allowed from: POSIX Thread and signal handler

This function relinquishes the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the `funlockfile()` function.

Usage

```
void funlockfile (FILE *file)
```

Arguments

- `file`
Pointer to the file stream.

Example

```
#include "services/stdio.h"

FILE *file;
file = fopen("test.txt", "r+b");

/* unlocks a file, counter decremented */
funlockfile(file);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc_unlocked](#)
[putc_unlocked](#) [<stdio.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

fwrite

Allowed from: POSIX Thread and signal handler

This function writes, from the array pointed to by ptr, up to nitems elements whose size is specified by size, to the stream pointed to by stream.

Usage

```
size_t fwrite (const void *ptr,  
               size_t      size,  
               size_t      nitems,  
               FILE         *stream)
```

Arguments

- ptr
String to write to the stream.
- size
Size of items to write in bytes.
- nitems
Number of items to write.
- stream
Pointer to the file stream.

Return Value

- size_t
On successful completion, return the number of items written.
- Number less than nitems
If the call fails, it returns a number less than nitems and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
char list[30];
int i,numwritten;

if ( (stream = fopen("stream1","w + t")) != NULL)
{
    for(i = 0;i < 25, i++)
        list[i] = (char) ('z' + i);

    /* Write 25 characters to the stream */
    numwritten = fwrite(list,sizeof(char),25,stream);

    fclose(stream);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	ferror
fopen	printf
putc	puts
write	<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

getc

Allowed from: POSIX Thread and signal handler

This function obtains the next byte as an unsigned character converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream.

Implementing this function as a macro might cause this function to evaluate the stream more than once. If used in this way, be sure that the argument is never an expression with side effects.

Usage

```
int getc (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- int
Next byte from the input stream pointed to by stream.
- EOF

If the call fails, errno is set to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;

/* Get one character from stdin */
ch = getc(stdin);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fgetc](#)

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

getc_unlocked

Allowed from: POSIX Thread and signal handler

This function obtains the next byte as an unsigned character converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream. getc_unlocked is not required to be implemented in a thread-safe manner.

Usage

```
int getc_unlocked (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

Return Value

- stream
Next byte from the input stream pointed to by stream.
- EOF

If the call fails, errno is set to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *file;
int ch;

/* Get one character from file */
ch = getc_unlocked('I', file);
```


Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc](#)
[getchar](#) [putc](#)
[putchar](#) [<stdio.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

getchar

Allowed from: POSIX Thread and signal handler

This function obtains the next byte as an unsigned character converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream.

Usage

```
int getchar (void)
```

Return Value

- stream

Next byte from the input stream pointed to by stream.

- EOF

If the call fails, errno is set to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;

/* Get a character from console */
ch = getchar();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc](#)

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

getchar_unlocked

Allowed from: POSIX Thread and signal handler

This function obtains the next byte as an unsigned character converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream. `getchar_unlocked` is not required to be implemented in a thread-safe manner.

Usage

```
int getchar_unlocked (void)
```

Return Value

- stream
Next byte from the input stream pointed to by stream.
- EOF

If the call fails, `errno` is set to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;

/* Get a character from console */
ch = getchar_unlocked();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getc
getchar	putc
putchar	<stdio.h>
Nucleus POSIX Run Time Library APIs	

getenv

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function searches the environment of the calling process for the environment variable name, if it exists, and returns a pointer to the value of the environment variable. This function does not modify the pointer to the environment variable.

Usage

```
char *getenv (const char *name)
```

Arguments

- name
Pointer to the name of the environment variable to be searched for.

Return Value

- char*
Pointer to a string containing the value for the specified name.
- NULL
The specified name cannot be found in the environment of the calling process.

Example

```
#include "services/stdlib.h"

static char *result;
...
result = getenv("PATH");
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [sem_post](#)
[sem_timedwait](#) [sem_trywait](#)
[sem_wait](#) [<semaphore.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

getpid

This function gets the process ID.

Usage

```
pid_t getpid(void)
```

Return Values

- Process ID
The ID of the process.

Example

```
#include "services/unistd.h"

pid_t pid;

/* Get the process Id */
pid = getpid();
```

Related Topics

[kill](#)

the Base Definitions volume of IEEE Std 1003.1-2001

[<sys/types.h>](#)

[<unistd.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

gets

Allowed from: POSIX Thread and signal handler

This function reads bytes from the standard input stream, stdin, into the array pointed to by s, until a <newline> is read or an end-of-file condition is encountered. Any <newline> is discarded and a null byte is placed immediately after the last byte read into the array.

Usage

```
char* gets (char *s)
```

Arguments

- s
Pointer to the buffer into which we have to read.

Return Value

- char*
On successful completion, returns a pointer to a string.
- NULL

If the call is unsuccessful, a null pointer is returned and errno is set to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOMEM

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

char string[100];

/* This will get the string from stdin */
/* User has to make sure that the length of the string entered on stdin
   should be less than the memory allocated for holding that string */

gets(string);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [feof](#)

[ferror](#)

[fgets](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

gmtime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function converts the time, in seconds, since the epoch pointed to by timer into a broken-down time, expressed as Coordinated Universal Time (UTC). This function is non-reentrant. For a re-entrant version of this function, refer to [gmtime_r](#).

Usage

```
struct tm* gmtime (const time_t *timer)
```

Arguments

- timer
Pointer to the time in seconds since epoch.

Return Value

- tm*
Pointer to the broken down structure if successful.
- NULL
An error is encountered.

Example

```
#include "services/time.h"

time_t timer;
struct tm *conv;

timer = 1234;

/* Find out the broken down time */
conv = gmtime(&timer);
```

Related Topics

asctime	Base Definitions volume of IEEE Std 1003.1-2001
clock	ctime
difftime	localtime
mktime	strftime
time	utime
<time.h>	Nucleus POSIX Run Time Library APIs
gmtime_r	

gmtime_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function converts the time, in seconds, since the epoch pointed to by timer into a broken-down time, expressed as Coordinated Universal Time (UTC). This is the re-entrant version of [gmtime](#).

Usage

```
struct tm *gmtime_r (const time_t *timer,  
                    struct tm      *result)
```

Arguments

- timer
Pointer to the time in seconds since epoch.
- result
To be filled with broken downtime

Return Value

- tm*
Pointer to the broken down structure if successful.
- NULL
An error is encountered.

Example

```
#include "services/time.h"  
  
time_t timer;  
struct tm *conv;  
struct tm result;  
  
timer = 1234;  
  
/* Find out the broken down time */  
conv = gmtime(&timer,&result);
```

Related Topics

[asctime](#)

[clock](#)

[difftime](#)

[mktime](#)

[time](#)

[<time.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[ctime](#)

[localtime](#)

[strftime](#)

[utime](#)

[Nucleus POSIX Run Time Library APIs](#)

hypot

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the value of the square root of x^2+y^2 without undue overflow or underflow.

Usage

```
double hypot (double x,  
              double y)
```

Arguments

- x
The x coordinate.
- y
The y coordinate.

Return Value

- double
Returns the length of the hypotenuse of a right-angled triangle with sides of length x and y.

Example

```
#include "services/math.h"  
  
static double x, y;  
static double result;  
...  
x = 3;  
y = 4;  
...  
result = hypot(x, y);  
...
```

Related Topics

[atan2](#)

Base Definitions volume of IEEE Std 1003.1-2001

[<math.h>](#)

[sqrt](#)

[Nucleus POSIX Run Time Library APIs](#)

imaxabs

This function returns the absolute value.

Usage

```
intmax_t imaxabs(intmax_t j)
```

Arguments

- `j`
The integer value.

Return Values

- `intmax_t`
The absolute integer value.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

imaxdiv

This function returns the quotient and remainder.

Usage

```
imaxdiv_t imaxdiv(intmax_t numer,  
                  intmax_t denom)
```

Arguments

- **numer**
The numerator.
- **denom**
The denominator.

Return Values

- **imaxdiv_t**
Structure containing the quotient and remainder.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

isalnum

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class alpha or digit in the program's current locale.

Usage

```
int isalnum (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not an alphanumeric character.
- Non zero
c is an alphanumeric character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isalnum(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

[isalpha](#)

[isctrl](#)

[isdigit](#)

[isgraph](#)

[islower](#)

[isprint](#)

[ispunct](#)

[isspace](#)

[isupper](#)

[isxdigit](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

isalpha

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class alpha in the program's current locale.

Usage

```
int isalpha (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not an alphabetic character.
- Non zero
c is an alphabetic character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isalpha(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<ctype.h>](#)

[isalnum](#)

[isctrl](#)

[isdigit](#)

[isgraph](#)

[islower](#)

[isprint](#)

[ispunct](#)

[isspace](#)

[isupper](#)

[isxdigit](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

isascii

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a 7-bit US-ASCII character code. The `isascii()` function is defined on all integer values.

Usage

```
int isascii (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a 7-bit US-ASCII character code.
- Non zero
c is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isascii(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

[Nucleus POSIX Run Time Library APIs](#)

isblank

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class blank in the program's current locale.

Usage

```
int isblank (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a <blank>.
- Non zero
c is a <blank>.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 32;
...
flag = isblank(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

[isalnum](#)

[isalpha](#)

[isctrl](#)

[isdigit](#)

[isgraph](#)

[islower](#)

[isprint](#)

[ispunct](#)

[isspace](#)

[isupper](#)

[isxdigit](#)

[Nucleus POSIX Run Time Library APIs](#)

isctrl

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class *cntrl* in the program's current locale.

Usage

```
int isctrl (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a control character.
- Non zero
c is a control character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isctrl(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isdigit	isgraph
islower	isprint
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

isdigit

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class digit in the program's current locale.

Usage

```
int isdigit (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a decimal digit.
- Non zero
c is a decimal digit.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 48;
...
flag = isdigit(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isctrl	isgraph
islower	isprint
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

isgraph

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class graph in the program's current locale.

Usage

```
int isgraph (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is a character with an invisible representation.
- Non zero
c is a character with a visible representation.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isgraph(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isctrl	isdigit
islower	isprint
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

islower

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class lower in the program's current locale.

Usage

```
int islower (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a lowercase letter.
- Non zero
c is a lowercase letter.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = islower(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isctrl	isdigit
isgraph	isprint
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

isprint

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class print in the program's current locale.

Usage

```
int isprint (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a printable character.
- Non zero
c is a printable character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isprint(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	<ctype.h>
isalnum	isalpha
isctrl	isdigit
isgraph	islower
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

ispunct

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class *punct* in the program's current locale.

Usage

```
int ispunct (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a punctuation character.
- Non zero
c is a punctuation character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/cttype.h"

static int target, flag;
...
target = 65;
...
flag = ispunct(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isctrl	isdigit
isgraph	islower
ispunct	isspace
isupper	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

isspace

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class space in the program's current locale.

Usage

```
int isspace (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a white space character.
- Non zero
c is a white space character.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 32;
...
flag = isspace(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

[isalnum](#)

[isalpha](#)

[isctrl](#)

[isdigit](#)

[isgraph](#)

[islower](#)

[isprint](#)

[ispunct](#)

[isupper](#)

[isxdigit](#)

[setlocale](#)

[Nucleus POSIX Run Time Library APIs](#)

isupper

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class upper in the program's current locale.

Usage

```
int isupper (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not an uppercase letter.
- Non zero
c is a uppercase letter.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isupper(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <ctype.h>

isalnum	isalpha
isctrl	isdigit
isgraph	islower
isprint	ispunct
isspace	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

isxdigit

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function tests whether *c* is a character of class *xdigit* in the program's current locale.

Usage

```
int isxdigit (int c)
```

Arguments

- *c*
The character to be tested.

Return Value

- 0
c is not a hexadecimal digit.
- Non zero
c is a hexadecimal digit.

Description

The function ensures that the input, *c*, is either an unsigned char or equal to the value of the EOF macro. If the argument has any other value, the behavior is undefined.

Example

```
#include "services/ctype.h"

static int target, flag;
...
target = 65;
...
flag = isxdigit(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	<ctype.h>
isalnum	isalpha
isctrl	isdigit
isgraph	islower
isprint	ispunct
isspace	isxdigit

[Nucleus POSIX Run Time Library APIs](#)

labs

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the absolute value of the long integer operand *i*.

Usage

```
long labs (long i)
```

Arguments

- *i*
Long integer operand.

Return Value

- long
Absolute value of the long integer operand.

Example

```
#include "services/stdlib.h"

static int target;
static long result;
...
target = 65;
...
result = labs(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [abs](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

ldexp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the quantity $x * 2^{\text{exp}}$.

Usage

```
double ldexp (double x,  
              int   exp)
```

Arguments

- **x**
The x argument.
- **exp**
The exponent value.

Return Value

- **double**
Returns x multiplied by 2, raised to the power exp.

Example

```
#include "services/math.h"  
  
static double target;  
static double result;  
...  
target = 3;  
...  
result = ldexp(target, 2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [frexp](#)

<math.h>

[Nucleus POSIX Run Time Library APIs](#)

ldiv

This function returns the quotient and remainder of a long division.

Usage

```
ldiv_t ldiv(long numer,  
            long denom)
```

Arguments

- **numer**
The numerator.
- **denom**
The denominator.

Return Values

- **ldiv_t**
Structure containing the quotient and remainder.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

llabs

This function returns a long integer absolute value.

Usage

```
long long llabs(long long i)
```

Arguments

- *i*
The long integer value.

Return Values

- long long
The absolute long integer value.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

lldiv

This function computes the quotient and remainder of a long division.

Usage

```
lldiv_t lldiv(long long numer,  
              long long denom)
```

Arguments

- **numer**
The numerator.
- **denom**
The denominator

Return Values

- **lldiv_t**
Structure that will contain the quotient and remainder.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

localtime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the time, in seconds, since the epoch pointed to by timer into a broken-down time, expressed as a local time. The function corrects the time zone and any seasonal time adjustments. This function is non-reentrant. For the reentrant version of this function, refer to [localtime_r](#).

Usage

```
struct tm* localtime (const time_t* timer)
```

Arguments

- timer
Pointer to the time in seconds since epoch.

Return Value

- tm*
Pointer to the broken-down time structure, returned upon success.
- NULL
An error is encountered.

Example

```
#include "services/time.h"

time_t    timer;
struct tm *conv;

timer = 1234;

/* Find out the broken down time */
conv = localtime(&timer);
```

Related Topics

asctime	Base Definitions volume of IEEE Std 1003.1-2001
clock	ctime
difftime	getdate
gmtime	mktime
strftime	time
<time.h>	tzset
utime	Nucleus POSIX Run Time Library APIs

localtime_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the time, in seconds, since the epoch pointed to by timer into a broken-down time, expressed as a local time. The function corrects the time zone and any seasonal time adjustments. This is the re-entrant version of [localtime](#).

Usage

```
struct tm *localtime_r (const time_t *timer,  
                        struct tm    *result)
```

Arguments

- timer
Pointer to the time in seconds since epoch.
- result
User provided structure of type tm* to be filled with the broken-down time.

Return Value

- tm*
Pointer to the broken-down time structure, returned upon success.
- NULL
An error is encountered.

Example

```
#include "services/time.h"  
  
time_t timer;  
struct tm *conv;  
struct tm result;  
  
timer = 1234;  
  
/* Find out the broken down time */  
conv = localtime(&timer,&result);
```

Related Topics

asctime	Base Definitions volume of IEEE Std 1003.1-2001
clock	ctime
difftime	getdate
gmtime	mktime
strftime	time
<time.h>	tzset

utime

Nucleus POSIX Run Time Library APIs

log

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the natural logarithm of its argument x , $\log(x)$.

Usage

```
double log (double x)
```

Arguments

- x
The x argument.

Return Value

- double
Returns the natural logarithm of x .

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 1;
...
result = log(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [exp](#)
[log10](#) [<math.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

log10

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the base 10 logarithm of their argument x, $\log_{10}(x)$.

Usage

```
double log10 (double x)
```

Arguments

- x
The x argument.

Return Value

- double
Returns the base 10 logarithm of x.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 1;
...
result = log10(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [log](#)
[pow](#) [<math.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

malloc

Allowed from: POSIX Thread and signal handler

This function allocates unused space for an object whose size, in bytes, is specified by size and whose value is unspecified.

Usage

```
void *malloc (size_t size)
```

Arguments

- size
Size of the required memory.

Return Value

- void*
Pointer to the allocated space.
- NULL
An error is encountered or size is zero. It sets errno to the corresponding error:
 ENOMEM
 Insufficient storage space is available.

Example

```
#include "services/stdlib.h"

static int *target;
...
target = (int *)malloc(sizeof(int));
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [calloc](#)

[free](#) [realloc](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

memchr

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function locates the first occurrence of *c* (converted to an unsigned char) in the initial *n* bytes (each interpreted as unsigned char) of the object pointed to by *s*.

Usage

```
void *memchr (const void *s,  
              int         c,  
              size_t      n)
```

Arguments

- *s*
Pointer to the target object.
- *c*
Character to search for.
- *n*
Number of initial bytes of *s* to search in.

Return Value

- `void *`
Pointer to the byte.
- `NULL`
Byte does not occur.

Example

```
#include "services/string.h"  
  
...  
static char string[6], *result;  
...  
  
strcpy(string, "abAde");  
result = (char *) memchr(string, 65, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <string.h>

[Nucleus POSIX Run Time Library APIs](#)

memcmp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function compare the first n bytes (each interpreted as unsigned char) of the object pointed to by s1 to the first n bytes of the object pointed to by s2.

Usage

```
int memcmp (const void *s1,
            const void *s2,
            size_t      n)
```

Arguments

- s1
First object.
- s2
Second object.
- n
Number bytes to compare.

Return Value

- > 0
s1 is greater than s2.
- 0
s1 is equal to s2.
- < 0
s1 is less than s2.

Example

```
#include "services/string.h"

...
static char string1[6], string2[6];
static int flag;
...

strcpy(string1, "abAde");
strcpy(string2, "abAde");
...

flag = memcmp(string1, string2, 4);

...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 `<string.h>`

[Nucleus POSIX Run Time Library APIs](#)

memcpy

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copy *c* bytes from the object pointed to by *s2* into the object pointed to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

Usage

```
void *memcpy (void      *s1,  
              const void *s2,  
              size_t     n)
```

Arguments

- *s1*
First object.
- *s2*
Second object.
- *n*
Number of bytes to copy.

Return Value

- *void**
Pointer to the value of the destination.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6], *result;  
...  
  
strcpy(string1, "-----");  
strcpy(string2, "abAde");  
...  
  
result = (char *) memcpy(string1, string2, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <string.h>

[Nucleus POSIX Run Time Library APIs](#)

memmove

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copy n bytes from the object pointed to by s2 into the object pointed to by s1.

Usage

```
void *memmove (void      *s1,  
               const void *s2,  
               size_t     n)
```

Arguments

- s1
First object.
- s2
Second object.
- n
Number of bytes to move.

Return Value

- void*
Pointer to the value of the destination.

Description

In this function, the copying takes place as if the n bytes from the object pointed to by s2 are first copied into a temporary array of n bytes that does not overlap the objects pointed to by s1 and s2, and then the n bytes from the temporary array are copied into the object pointed to by s1.

Example

```
#include "services/string.h"  
...  
static char string1[6], string2[6], *result;  
...  
strcpy(string1, "-----");  
strcpy(string2, "abAde");  
...  
result = (char *) memmove(string1, string2, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <string.h>

[Nucleus POSIX Run Time Library APIs](#)

memset

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copies *c* (converted to an unsigned char) into each of the first *n* bytes of the object pointed to by *s*.

Usage

```
void *memset (void    *s,  
              int      c,  
              size_t  n)
```

Arguments

- *s*
Target object.
- *c*
Character to set.
- *n*
Number of initial bytes of *s* to set in.

Return Value

- *s*
Pointer to the value of the destination.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], *result;  
...  
  
strcpy(string1, "-----");  
...  
  
result = (char *) memset(string1, '*', 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <string.h>

[Nucleus POSIX Run Time Library APIs](#)

mktime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function converts the broken-down time, expressed as local time, in the structure pointed to by `timeptr`, into a time since the epoch value.

Usage

```
time_t mktime (struct tm *timeptr)
```

Arguments

- `timeptr`
Pointer to the `tm` structure

Return Value

- `time_t`
Specified time since the epoch encoded as a value of type `time_t`.
- `-1`
An error is encountered.

Example

```
#include "services/time.h"

struct tm time_str;
time_t conv_time;

time_str.tm_year = 2001;
time_str.tm_month = 7;
time_str.tm_mday = 4;
time_str.tm_hour = 0;
time_str.tm_min = 0;
time_str.tm_sec = 1;
time_str.tm_isdst = -1;

/* Convert the broken down time in to a time since Epoch */
conv_time = mktime(&time_str);
```

Related Topics

[asctime](#)

[clock](#)

[difftime](#)

[localtime](#)

[time](#)

[tzset](#)

Base Definitions volume of IEEE Std 1003.1-2001

[ctime](#)

[gmtime](#)

[strftime](#)

[<time.h>](#)

[utime](#)

[utime](#)

[Nucleus POSIX Run Time Library APIs](#)

modf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function breaks the argument *x* into integral and fractional parts, each of which has the same sign as the argument.

Usage

```
double modf (double x,  
             double *iptr)
```

Arguments

- *x*
The *x* argument to be broken up into integral and fractional parts.
- *iptr*
Pointer to the integral part as double.

Return Value

- double
Returns the signed fractional part of *x*.

Example

```
static double target, iptr;  
static double result;  
...  
target = 7.8921;  
...  
result = modf(target, &iptr);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [frexp](#)

[ldexp](#) [<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

mq_timedsend

This function attempts to send a message to a message queue in a time-limited manner (time-out allowed).

Usage

```
int mq_timedsend(mqd_t          mqdes,  
                 const char     *msg_ptr,  
                 size_t         msg_len,  
                 unsigned       msg_prio,  
                 const struct timespec *abs_timeout)
```

Arguments

- **mqdes**
Message queue.
- **msg_ptr**
Pointer to message.
- **msg_len**
Length of the message in bytes.
- **msg_prio**
Priority of the message.
- **abs_timeout**
Timeout value for the operation.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
No message is enqueued. The functions return -1 and sets `errno` to the corresponding error:
 - EAGAIN**
The `O_NONBLOCK` flag is set in the message queue description associated with `mqdes`, and the specified message queue is full.
 - EBADF**
The `mqdes` argument is not a valid message queue descriptor open for writing.
 - EINTR**
A signal interrupted the call to `mq_timedsend()`.
 - EINVAL**
The value of `msg_prio` was outside the valid range.

EINVAL

The process or thread would have blocked, and the `abs_timeout` parameter specified a nanoseconds field value less than zero, or greater than or equal to 1000 million.

EMSGSIZE

The specified message length, `msg_len`, exceeds the message size attribute of the message queue.

ETIMEDOUT

The `O_NONBLOCK` flag was not set when the message queue was opened, but the timeout expired before the message could be added to the queue.

Example

```
#include "services/pthread.h"
#include "services/unistd.h"
#include "services/time.h"
#include "services/signal.h"
#include "services/mqueue.h"
#include "services/sys/stat.h"
#include "services/stdio.h"

/* Global Variables */
static int Posix_Error = 1;
static mqd_t Psx_Mq_Timed_Send_Queue;
static unsigned long Task_2_messages_received;
static unsigned long Task_1_messages_sent;

/* Assuming the Message Queue has been previously created and
   Initialized */
...

/* Two threads will be used to send few messages back and forth until few
   iterations. The messagequeue send will be tested for timed operation by
   filling up the message queue.*/

/* This thread will receive message until 50 then stops. */
static void* Test_Psx_Thread1_Entry(void *argc)
{
    int status;
    unsigned long Receive_Message;
    unsigned message_prio;

    Task_2_messages_received = 0;

    /* Sleep for 2 seconds and let thread 2 sends all the message until the
       message queue is full */
    sleep(2);

    while(1)
    {
        /* Retrieve a message from Psx_Mq_Timed_Send_Queue, which thread 1
           writes to.
           Note that if the source queue is empty this task suspends until
           something becomes available. */
```

```
        status = mq_receive(Psx_Mq_Timed_Send_Queue,
                            (char*)&Receive_Message,
                            sizeof(unsigned long), &message_prio);

        /* Determine if the message was received successfully. */
        if (status >= 0)
        {
            Task_2_messages_received++;
        }
        else
        {
            printf("error error count: %i,
                    line: %i \n", Posix_Error++, __LINE__);
        }
        /* We will stop the test after few iterations */
        if (Task_2_messages_received == TEST_MQ_RECEIVE)
        {
            break;
        }
    }
    /* let thread main run */
    sleep(1);

    pthread_exit(NULL);

    return 0;
}

/* This thread will test the message queue timed send. The test will stop
after few iterations.*/
static void* Test_Psx_Thread2_Entry(void *argc)
{
    int status;
    unsigned long Send_Message;
    unsigned message_prio;
    struct timespec abs_timeout;

    /* Initialize the message contents. The receiver will examine the
    message contents for errors. */
    Send_Message = 0;

    /* Set the message priority to 10 */
    message_prio = 10;

    abs_timeout.tv_sec = 2;
    abs_timeout.tv_nsec = 0;

    while(1)
    {
        /* Send the message to Psx_Mq_Timed_Send_Queue, which thread 2 reads
        from. Note that if the destination queue fills up this task will
        timed suspend until room becomes available. */
        status = mq_timedsend(Psx_Mq_Timed_Send_Queue,
                              (char *)&Send_Message, sizeof(unsigned long),
                              message_prio, &abs_timeout);
    }
}
```



```
/* Determine if the message was sent successfully. */
if (status == 0)
{
    Task_1_messages_sent++;
}
else
{
    /* At 10 messages, mq_timedsend() will run out of buffer,
       timeout and error will be returned. */
    if (Task_1_messages_sent != 10)
    {
        printf("error error count: %i,
               line: %i \n", Posix_Error++, __LINE__);
    }
}

/* We will stop the test after few iterations */
if (Task_1_messages_sent == TEST_MQ_RECEIVE)
{
    break;
}

/* Modify the contents of the next message to send. */
Send_Message++;
}

/* let thread main run */
sleep(1);

pthread_exit(NULL);

return 0;
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	mq_open
mq_receive	mq_setattr
mq_timedreceive	time
<time.h>	<mqueue.h>

[Nucleus POSIX Run Time Library APIs](#)

pathconf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function determines the current value of a configurable limit or option (variable) associated with a file or directory.

Usage

```
long pathconf (const char *path,  
               int         name)
```

Arguments

- **path**
Points to a pathname of a file or directory.
- **name**
Variable to be queried relative to that file or directory.

Return Value

- **POSIX_ERROR**
Error encountered.
- **long**
Configurable limit associated with the file or directory on success.
- **EINVAL**
The value of name is invalid.

Example

```
#include "services/unistd.h"  
  
long value;  
  
/* Find the Maximum Name Length limit associated with the file  
   "/usr/file1.txt" */  
value = pathconf("/usr/file1.txt",NAME_MAX);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	confstr
<unistd.h>	<limits.h>
Shell and Utilities volume of IEEE Std 1003.1-2001	sysconf

perror

Allowed from: POSIX Thread and signal handler

This function maps the error number accessed through the symbol `errno` to a language-dependent error message.

Usage

```
void perror (const char *s)
```

Arguments

- `s`
Pointer to the error string.

Example

```
#include "services/stdio.h"
#include "services/stdlib.h"

char    *bufptr;
size_t  szbuf;

if ((bufptr = malloc(szbuf)) == NULL)
{
    perror("malloc");
    exit(2);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	confstr
<unistd.h>	<limits.h>
Shell and Utilities volume of IEEE Std 1003.1-2001	sysconf

[Nucleus POSIX Run Time Library APIs](#)

pow

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the value of x raised to the y power. If x is negative, the application ensures that y is an integer value.

Usage

```
double pow (double x,  
            double y)
```

Arguments

- x
The x argument.
- y
The y argument.

Return Value

- double
Returns the value of x raised to the y power.

Example

```
#include "services/math.h"  
  
static double x, y;  
static double result;  
...  
x = 3;  
y = 2;  
...  
result = pow(x, y);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [exp](#)

<math.h>

[Nucleus POSIX Run Time Library APIs](#)

printf

Allowed from: POSIX Thread and signal handler

This function places an output on the standard output stream stdout.

Usage

```
int printf (const char *format, ...)
```

Arguments

- format
Output formatting string.

Supported Flag Characters

- -
The result of the conversion is left justified within the field.
- 0
For d, i, o, u, x, X, , e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width.

Supported Length Modifiers

- l
Specifies that a following d,i,o,u, x, or X conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- d,i
The int argument will be converted to a signed decimal in the style “[]dddd”.
- o
The unsigned argument will be converted to unsigned octal format in the style “dddd”.
- u
The unsigned argument will be converted to unsigned decimal format in the style “dddd”
- x
The unsigned argument will be converted to unsigned hexadecimal format in the style “dddd”
- X
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.
- f,F
The double argument will be converted to decimal notation in the style “[]ddd.ddd”.

- e,E
The double argument will be converted in the style “[]d.ddde±dd”.
- g,G
The double argument will be converted in the style f or e (or in the style F or E in the case of a G conversion specifier).
- c
The int argument will be converted to an unsigned char, and the resulting byte will be written.
- s
The argument will be a pointer to an array of char. Bytes from the array will be written up to (but not including) any terminating null byte.
- %
Print a ‘%’ character; no argument is converted. The complete conversion specification will be%%.

Return Value

- int
On successful completion returns the number of bytes transmitted, otherwise.
- Negative Value
If the call fails, it returns a negative value and sets errno to the corresponding error:
 - EBADF
The file descriptor-underlying stream is not valid.
 - EIO
I/O error.
 - ENOSPC
Insufficient storage space is available.
 - EACCES
Permission denied.
 - EINVAL
The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int count = -1234;
```

```
/* Display Integers */
printf("Integer Formats :\n \tDecimal: %d Justified: % .6d\n\n\tUnsigned: %u \n",count,count,count);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fputc](#)

fscanf

<stdio.h>

<wchar.h>

Nucleus POSIX Run Time Library APIs

pthread_atfork

This function register fork handlers.

Note



The pthread_atfork() API is not going to be implemented, but is included in the Nucleus POSIX library for compliance with POSIX specification PSE-52 IEEE 1003.13-2003.

Usage

```
int pthread_atfork(void (*prepare)(void),
                  void (*parent)(void),
                  void (*child)(void))
```

Arguments

- **prepare**
This handler is called before fork() processing commences.
- **parent**
This handler is called after fork() processing completes in the parent process.
- **child**
This handler is called after fork() processing completes in the child process.

Return Values

- **0**
The function completed successfully.
- **ENOMEM**
Insufficient table space exists to record the fork handler addresses. The pthread_atfork() function does not return an error code of EINTR.

Example

```
#include "services/pthread.h"
void prepare(void){}
void parent(void){}
void child(void){}

int status;

/* Calling pthread_atfork() API */
status = pthread_atfork(prepare, parent, child);
if (status == POSIX_SUCCESS)
{
    printf("success \n");
}
```


Related Topics

The Base Definitions volume of IEEE Std 1003.1-2001 <sys/types.h>

[Nucleus POSIX Run Time Library APIs](#)

pthread_getcpuclockid

This function accesses a thread CPU-time clock.

Usage

```
int pthread_getcpuclockid(pthread_t thread_id,  
                           clockid_t *clock_id)
```

Arguments

- **thread_id**
The thread.
- **clock_id**
Pointer the clock ID of the CPU-time clock of the thread.

Return Values

- **POSIX_SUCCESS**
The function completed successfully.
- **ESRCH**
The value specified by **thread_id** does not refer to an existing thread.
- **EINVAL**
The value of the **clock_id** argument is an invalid.

Example

```
#include "services/pthread.h"  
#include "services/unistd.h"  
#include "services/time.h"  
#include "services/signal.h"  
#include "services/stdio.h"  
  
/* Global Variable */  
static int Posix_Error = 1;  
  
/*Testing getting the cpu clock id for this thread.*/  
  
static void* Test_Psx_Thread1_Entry(void *argc)  
{  
    int      status;  
    clockid_t clocktype;  
  
    status = pthread_getcpuclockid(Test_Psx_Thread1,&clocktype);  
    if (status == POSIX_SUCCESS)  
    {  
        if (clocktype != CLOCK_REALTIME)  
        {  
            printf("error error count: %i,  
                  line: %i \n",Posix_Error++,__LINE__);  
        }  
    }  
}
```

```
    else
    {
        printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);
    }

    pthread_exit(NULL);

    return 0;
}
```

Related Topics

[clock_getcpuclockid](#)

[clock_getres](#)

[timer_create](#)

[the Base Definitions volume of IEEE Std 1003.1-2001](#)

[<pthread.h>](#)

[<time.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

pthread_mutex_timedlock

This function attempts to lock a mutex in a time-limited manner (time-out allowed).

Usage

```
int pthread_mutex_timedlock(pthread_mutex_t *restrict      mutex,  
                           const struct timespec *restrict abs_timeout)
```

Arguments

- **mutex**
The mutex.
- **abs_timeout**
The timeout value.

Return Values

- **POSIX_SUCCESS**
The function completed successfully.
- **EOF**

If the function does not complete successfully, it sets **errno** to the corresponding error code:

EINVAL

The mutex was created with the protocol attribute having the value **PTHREAD_PRIO_PROTECT** and the calling thread's priority higher than the mutex' current priority ceiling.

EINVAL

The process or thread would have blocked. Also, the **abs_timeout** parameter specified a nanoseconds field value less than zero, or greater than or equal to 1000 million.

ETIMEDOUT

The mutex could not be locked before the specified timeout expired.

EAGAIN

The mutex could not be acquired because the maximum number of recursive locks for mutex has been exceeded.

EDEADLK

A deadlock condition was detected or the current thread already owns the mutex. This function does not return an error code of **EINTR**.

Example

```
#include "services/pthread.h"  
#include "services/unistd.h"  
#include "services/time.h"  
#include "services/signal.h"  
#include "services/errno.h"
```

```
#include "services/stdio.h"

/* Global Variables */
static int ptm_timed_cntr = 1;
static int Posix_Error    = 1;
static pthread_mutex_t ptm_timed_lock;

/* Both Threads are created with different priorities */
#define PSX_TEST_THREAD1_PRIORITY 120
#define PSX_TEST_THREAD2_PRIORITY 180

/* Assuming the mutex has been previously created and initialized */

/* This thread will lock the mutex and wait until the timeout
   has passed. */
static void* Test_Psx_Thread2_Entry(void *argc)
{
    int status;
    struct timespec abs_timeout;

    abs_timeout.tv_sec  = 3;
    abs_timeout.tv_nsec = 0;

    /* Time lock the mutex. This mutex will timeout. */
    status = pthread_mutex_timedlock(&ptm_timed_lock,&abs_timeout);
    if (status != ETIMEDOUT)
    {
        printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);
    }

    /* Test to see if thread 1 still has the mutex. It should block here. */
    status = pthread_mutex_lock(&ptm_timed_lock);
    if (status == POSIX_SUCCESS)
    {
        ptm_timed_cntr--;
    }
    else
    {
        printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);
    }

    /* Kill myself */
    pthread_exit(NULL);

    return(0);
}

/* This thread will own the lock and get the inherit thread 2 priority
   due to priority inheritance and bump it down when the thread is
   unlocked. */
static void* Test_Psx_Thread1_Entry(void *argc)
{
    int status;

    /* Locking the mutex */
    status = pthread_mutex_lock(&ptm_timed_lock);
    if (status == POSIX_SUCCESS)
    {

```

```
        sleep(1);
    }

    /* This thread priority will be bump to be the same as thread 2. This
       thread is still in critical section. */

    /* Wait for timeout */
    sleep(3);

    /* unlock and bump the priority down */
    status = pthread_mutex_unlock(&ptm_timed_lock);

    /* thread 1 should be instantaneously switching to thread 2 since the
       priority has been bumped down */
    if (ptm_timed_cntr != 0)
    {
        printf("error error count: %i, line: %i \n", Posix_Error++, __LINE__);
    }

    /* Kill myself */
    pthread_exit(NULL);

    return(NULL);
}
```

Related Topics

[pthread_mutex_destroy](#)

[pthread_mutex_trylock](#)

[the Base Definitions volume of IEEE Std 1003.1-2001](#)

[<time.h>](#)

[pthread_mutex_lock](#)

[time](#)

[<pthread.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

pthread_mutexattr_gettype

This function gets the mutex type attribute, attr, and stores in the object referenced by type.

Usage

```
int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
                              int *restrict type)
```

Arguments

- attr
The attribute structure.
- type
Pointer to return the mutex type.

Return Values

- POSIX_SUCCESS
The function completed successfully.
- EINVAL
The value specified by attr is invalid.

Example

```
#include "services/pthread.h"
#include "services/unistd.h"
#include "services/time.h"
#include "services/signal.h"
#include "services/stdio.h"

/* Global Variable */
static int Posix_Error = 1;
static pthread_t Test_Psx_Thread1;

/* This thread will test get mutex type. */
status = pthread_mutexattr_gettype(&matr, &mtx_type);
if (status == POSIX_SUCCESS)
{
    if (mtx_type != PTHREAD_MUTEX_ERRORCHECK)
    {
        printf("error error count: %i, line: %i \n", Posix_Error++, __LINE__);
    }
}
else
{
    printf("error error count: %i, line: %i \n", Posix_Error++, __LINE__);
}
```

Related Topics

pthread_cond_timedwait the Base Definitions volume of IEEE Std 1003.1-2001

<pthread.h>

Nucleus POSIX Run Time Library APIs

pthread_mutexattr_settype

This function sets the mutex type attribute.

Usage

```
int pthread_mutexattr_settype (pthread_mutexattr_t *attr,  
                               int type)
```

Arguments

- attr
The attribute structure.
- type
The new mutex type.

Return Values

- POSIX_SUCCESS
The function completed successfully.
- EINVAL
The value specified by attr is invalid.

Example

```
#include "services/pthread.h"  
#include "services/unistd.h"  
#include "services/time.h"  
#include "services/signal.h"  
#include "services/stdio.h"  
  
/* Global Variable */  
static int      Posix_Error = 1;  
static pthread_t Test_Psx_Thread1;  
  
/* This thread will test set mutex type.*/  
static void* Test_Psx_Thread1_Entry(void *argc)  
{  
    int status;  
    pthread_mutexattr_t mattr;  
    pthread_mutex_t      mtx;  
    int                  mtx_type;  
  
    status = pthread_mutexattr_init(&mattr);  
    if (status == POSIX_SUCCESS)  
    {  
        status = pthread_mutex_init(&mtx, &mattr);  
        if (status == POSIX_SUCCESS)  
        {  
            status =  
                pthread_mutexattr_settype(&mattr, PTHREAD_MUTEX_RECURSIVE);  
            if (status == POSIX_SUCCESS)  
            {  
                Test_Psx_Thread1 = pthread_create(&Test_Psx_Thread1, &mattr,  
                                                    Test_Psx_Thread1_Entry, &argc);  
            }  
        }  
    }  
}
```

```
        printf("error error count: %i,\n",Posix_Error++,__LINE__);\n    }\n    status =\n        pthread_mutexattr_settype(&mutexattr,PTHREAD_MUTEX_ERRORCHECK);\n    if (status == POSIX_SUCCESS)\n    {\n        Printf("success \n");\n    }\n    else\n    {\n        printf("error error count: %i,\n",Posix_Error++,__LINE__);\n    }\n}\nelse\n{\n    printf("error error count: %i,\n",Posix_Error++,__LINE__);\n}\n}\nelse\n{\n    printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);\n}
```

Related Topics

pthread_cond_timedwait
<pthread.h>

the Base Definitions volume of IEEE Std 1003.1-2001
[Nucleus POSIX Run Time Library APIs](#)

putc

Allowed from: POSIX Thread and signal handler

This function writes the byte specified by *c* (converted to an unsigned char) to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately.

Implementing this function as a macro might cause this function to evaluate the stream more than once. If used in this way, be sure that the argument is never an expression with side effects.

Usage

```
int putc (int  c,
          FILE *stream)
```

Arguments

- *c*
Pointer to the byte to be written.
- *stream*
Pointer to the file stream.

Return Value

- *int*
On successful completion, returns the value it has written.
- EOF
If the call fails, it returns EOF and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;
```

```
/* Put a character 'F' on stdout */  
ch = putc('F',stdout);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fputc](#)

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

putc_unlocked

Allowed from: POSIX Thread and signal handler

This function writes the byte specified by *c* (converted to an unsigned char) to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. *putc_unlocked* is not required to be implemented in a thread-safe manner.

Usage

```
int putc_unlocked (int c,
                  FILE *stream)
```

Arguments

- *c*
Pointer to the byte to be written.
- *stream*
File stream.

Return Value

- *int*
On successful completion, returns the value it has written.
- EOF

If the call fails, it returns EOF and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *file;
int ch;

/* sends a char to file */
```

```
ch = putc_unlocked('M', file);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[getc](#)

[getchar](#)

[putc](#)

[putchar](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

putchar

Allowed from: POSIX Thread and signal handler

This function writes the byte specified by *c* (converted to an unsigned char) to the standard output, and advances the indicator appropriately. This is equivalent to [putc\(c, stdout\)](#).

Usage

```
int putchar (int c)
```

Arguments

- *c*
Specifies the byte to be written.

Return Value

- *int*
On successful completion, returns the value it has written.
- EOF
If the call fails, it returns EOF and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ret;

/* Put a character 'F' on stdout */
ret = putchar('F');
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<stdio.h>

[putc](#)

[Nucleus POSIX Run Time Library
APIs](#)

putchar_unlocked

Allowed from: POSIX Thread and signal handler

This function writes the byte specified by *c* (converted to an unsigned char) to the standard output, and advances the indicator appropriately. `putchar_unlocked` is not required to be implemented in a thread-safe manner.

Usage

```
int putchar_unlocked (int c)
```

Arguments

- *c*
Specifies the byte to be written.

Return Value

- `int`
On successful completion, returns the value it has written.
- `EOF`
If the call fails, it returns `EOF` and sets `errno` to the corresponding error:

`EBADF`

The file descriptor-underlying stream is not valid.

`EIO`

I/O error.

`ENOSPC`

Insufficient storage space is available.

`EACCES`

Permission denied.

`EINVAL`

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

int ch;

/* Put a character 'F' on stdout */
ch = putchar('F');
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc](#)

[getchar](#)

[putchar](#)

[Nucleus POSIX Run Time Library APIs](#)

[putc](#)

[<stdio.h>](#)

puts

Allowed from: POSIX Thread and signal handler

This function writes the string pointed to by *s*, followed by a <newline>, to the standard output stream stdout. The terminating null byte is not written.

Usage

```
int puts (const char *s)
```

Arguments

- *s*
Specifies the string to be written.

Return Value

- *int*
On successful completion, returns a non-zero number.
- EOF
If the call fails, it returns EOF and sets *errno* to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/time.h"
#include "services/stdio.h"

time_t now;
int minutes_to_event;

time(&now);
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",minutes_to_event);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fopen](#)

[fputs](#)

[putc](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

qsort

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function sorts an array of `nel` objects, the initial element of which is pointed to by `base`.

Usage

```
void qsort (void    *base,
            size_t  nel,
            size_t  width,
            int      (*compar)(const void *, const void *))
```

Arguments

- **base**
Pointer to the base of the data.
- **nel**
Number of elements.
- **width**
Size of the elements in bytes.
- **compar**
Pointer to a comparison routine which takes two pointers to items and returns an integer `<`, `=`, or `>` 0 according to as item one is `<`, `=`, or `>` item two.

Example

```
#include "services/stdlib.h"

int compar(const void *x, const void *y)
{
    long a, b;
    int t;

    a = (long) (*(long*)x);
    b = (long) (*(long*)y);

    if (a < b)
        t = -1;
    else if (a == b)
        t = 0;
    else
        t = 1;

    return(t);
}

int main(int argc, char* argv[])
{
    long a[1000];
    long *b[1000];
    long *p;
    int i;
```

```
/* Fill up the array b with random numbers. */
for(i = 0; i < 1000; i++)
{
    a[i] = (long)rand();
    b[i] = &a[i];
}

/* Now apply quick sort algorithm over array b */
qsort(&b[1000],1000,4,compar);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[Nucleus POSIX Run Time Library APIs](#)

rand

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes a sequence of pseudo-random integers in the $[0, \{\text{RAND_MAX}\}]$ range with a period of at least 2^{32} . This function is non-reentrant. Refer to [rand_r](#) for the reentrant version of this function.

Usage

```
int rand (void)
```

Return Value

- **int**
Next pseudo-random number in the sequence.

Example

```
#include "services/stdlib.h"

static int result;
...
result = rand();
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <stdlib.h>

[Nucleus POSIX Run Time Library APIs](#)

rand_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes a sequence of pseudo-random integers in the $[0, \{RAND_MAX\}]$ range with a period of at least 2^{32} . This is the re-entrant version of [rand](#).

Usage

```
int rand_r (unsigned *seed)
```

Arguments

- **seed**
Pointer to the seed for random number generation.

Return Value

- **int**
Next pseudo-random number in the sequence.

Example

```
#include "services/stdlib.h"

static int result;
unsigned seed;

seed = 12345;
...
result = rand_r(&seed);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

realloc

Allowed from: POSIX Thread and signal handler

This function changes the size of the memory object pointed to by ptr to the size specified by size.

Usage

```
void *realloc (void *ptr,  
               size_t size)
```

Arguments

- ptr
Pointer to the previous memory object.
- size
Size of the memory reallocated.

Return Value

- void*
Pointer to the allocated space.
- NULL
An error is encountered or size is 0. It sets errno to the following error code:
 ENOMEM
 Insufficient storage space is available.

Description

The content of the object remains unchanged up to the lesser of the new and old sizes. If the new size of the memory object requires movement of the object, the space for the previous instantiation of the object is freed. If the new size is larger, the contents of the newly allocated portion of the object are unspecified. If size is 0 and ptr is not a null pointer, the object pointed to is freed. If the space cannot be allocated, the object remains unchanged.

Example

```
#include "services/stdlib.h"  
  
static int *target;  
...  
target = (int *) malloc(sizeof(int));  
...  
realloc(target, sizeof(int)*5);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [calloc](#)

[free](#)

[<stdlib.h>](#)

[malloc](#)

[Nucleus POSIX Run Time Library APIs](#)

remove

Allowed from: POSIX Thread and signal handler

This function makes the file, pointed to by path, no longer accessible by that name.

Usage

```
int remove (const char *path)
```

Arguments

- path
Path to the file to be deleted.

Return Value

- 0
Function call successful.

- -1
If the call fails, it returns -1 and sets errno to the corresponding error:

ENOENT

A component of path does not name an existing file, or the path argument names a nonexistent directory or points to an empty string.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

Description

If path does not name a directory, remove(path) is equivalent to [unlink\(path\)](#).

If path names a directory, remove(path) is equivalent to [rmdir\(path\)](#).

Example

```
#include "services/sys/stat.h"
#include "services/stdio.h"

int ret;

mkdir("templ", S_IRWX);
ret = remove("templ");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [rmdir](#)
[unlink](#) [<stdio.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

rewind

Allowed from: POSIX Thread and signal handler

This function resets the file position indicator in a stream.

Usage

```
void rewind (FILE *stream)
```

Arguments

- stream
Pointer to the file stream.

errno Values

- EBADF
The file descriptor-underlying stream is not valid.
- EIO
I/O error.
- ENOSPC
Insufficient storage space is available.
- EACCES
Permission denied.
- EINVAL
The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;

if ( (stream = fopen("stream1", "w+")) != NULL)
{
    rewind(stream);
    fclose(stream);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fseek](#)

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

scanf

Allowed from: POSIX Thread and signal handler

This function reads from the standard input stream stdin.

Usage

```
int scanf (const char *format, ... )
```

Arguments

- format
Input formatting string.

Supported Length Modifiers

- l
Specifies that a following d,i,o,u, x,orX conversion specifier applies to a long or unsigned long argument.
- h
Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short.

Supported Conversion Specifiers

- d
Matches an optionally signed decimal integer.
- i
Matches an optionally signed integer.
- o
Matches an optionally signed octal integer.
- u
Matches an optionally signed decimal integer.
- x
Matches an optionally signed hexadecimal integer.
- a,e,f,g
Matches an optionally signed floating-point number.
- s
Matches a sequence of bytes that are not white-space characters.
- [
Matches a non-empty sequence of bytes from a set of expected bytes (the scanset).

- **c**
Matches a sequence of bytes of the number specified by the field width.
- **%**
Matches a single ‘%’ character; no conversion or assignment occurs. The complete conversion specification is%%.

Return Value

- **int**
Number of successfully matched and assigned input items.
- **EOF**
If error encountered.

Example

```
int i;  
float x;  
char name[50];  
  
(void) scanf("%2d%f%*d %[0123456789]", &i, &x, name);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getc
printf	strtod
strtol	strtoul
<stdio.h>	<wchar.h>

[Nucleus POSIX Run Time Library APIs](#)

setbuf

Allowed from: POSIX Thread and signal handler

This function assigns buffering to the stream.

Usage

```
void setbuf (FILE *stream,  
            char *buf)
```

Arguments

- **stream**
Pointer to the file stream.
- **buf**
Array to be used as buffer.

errno Values

- **EBADF**
The file descriptor-underlying stream is not valid.
- **EIO**
I/O error.
- **ENOSPC**
Insufficient storage space is available.
- **EACCES**
Permission denied.
- **EINVAL**
The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"  
  
FILE *stream;  
char buf[BUFSIZ];  
  
if ( (stream = fopen("stream1", "a")) != NULL)  
{  
    setbuf(stream1, buf);  
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[setvbuf](#)

[fopen](#)

[<stdio.h>](#)

Nucleus POSIX Run Time Library APIs

setenv

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function updates or adds a variable in the environment of the calling process.

Usage

```
int setenv (const char *envname,  
            const char *envval,  
            int         overwrite)
```

Arguments

- **envname**
Points to a string containing the name of an environment variable to be added or altered.
- **envval**
Environment variable set to the value envval points to.
- **overwrite**
Flag if true indicates that the environment variable must be overwritten if it already exists.

Return Value

- **POSIX_SUCCESS**
Successful completion.
- **POSIX_ERROR**
Indicates failure of the service and sets errno to the corresponding error code:
 - EINVAL**
The name argument is a null pointer, points to an empty string, or points to a string containing an '=' character.
 - ENOMEM**
Insufficient memory was available to add a variable or its value to the environment.

Example

```
#include "services/stdlib.h"  
  
...  
setenv("PATH", "c:\\bin" );  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<stdlib.h>

[unsetenv](#)

[getenv](#)

<sys/types.h>

<unistd.h>

Nucleus POSIX Run Time Library APIs

setlocale

This function sets the program locale.

Usage

```
#include <locale.h>
...
char *setlocale(int      category,
                const char *locale)
```

Arguments

- **category**
Indicates the category of the locale information that will be set.
- **locale**
Pointer to a character string containing the required setting of category.

Return Values

- **char ***
Pointer to the string associated with the specified category for the new locale. Otherwise NULL.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

setvbuf

Allowed from: POSIX Thread and signal handler

This function assigns buffering to the stream.

Usage

```
int setvbuf (FILE    *stream,
             char     *buf,
             int       type,
             size_t    size)
```

Arguments

- **stream**
Pointer to the file stream.
- **buf**
Array to be used as buffer.
- **type**
Determines how stream will be buffered. This can be set to one of the following:
 - _IOFBF**
Causes Input/Output to be fully buffered.
 - _IOLBF**
Causes Input/Output to be line buffered.
 - _IONBF**
Causes Input/Output to be unbuffered.
- **size**
Specifies the size of the array.

Return Value

- **0**
Function call successful.
- **Non-zero**
If the call fails, it returns a non-zero value and sets `errno` to the corresponding error:
 - EBADF**
The file descriptor-underlying stream is not valid.
 - EIO**
I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

char buf[1024];
FILE *stream;

if ( (stream = fopen("stream1", "a")) != NULL)
{
    setvbuf(stream, buf, _IOFBF, sizeof(buf));
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fopen](#)
[setbuf](#) [<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

signal

This function registers a signal handler for a signal.

Usage

```
void (*signal(int sig,  
              void (*func)(int)))(int)
```

Arguments

- **sig**
The signal number.
- **func**
Function pointer to signal handler function. This can be SIG_DFL or SIG_IGN.

Return Values

- **func**
If the service completes successfully, the value of func for the most recent signal call is returned.
- **SIG_ERR**
Indicates that the service fails and sets errno to the corresponding error code:

EINVAL

The sig argument is not a valid signal number, or an attempt is made to catch a signal that cannot be caught, or ignore a signal that cannot be ignored.

Example

```
#include "services/signal.h"  
  
/* Called from Thread context */  
int status;  
  
/* Set up SIGUSR2 signal as signal default */  
if (signal(SIGUSR2,SIG_DFL) != SIG_ERR)  
{  
    /* Set up SIGUSR1 signal as signal ignore */  
    if (signal(SIGUSR1,Test_Psx_Signal_Handler) != SIG_ERR)  
    {  
        /* Initialize the thread attributes */  
        status = pthread_attr_init(&attr);  
    }  
    else  
    {  
        printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);  
    }  
}  
else  
{  
    printf("error error count: %i, line: %i \n",Posix_Error++,__LINE__);  
}
```

Related Topics

pause

sigsuspend

<signal.h>

sigaction

the Base Definitions volume of IEEE Std 1003.1-2001

[Nucleus POSIX Run Time Library APIs](#)

sin

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the sine of the argument *x*, measured in radians.

Usage

```
double sin (double x)
```

Arguments

- *x*
The argument with which to calculate the sine.

Return Value

- `double`
Returns the sine of *x*.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 3.14159;
...
result = sin(target);
...
```

Related Topics

[asin](#)

[<math.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[Nucleus POSIX Run Time Library APIs](#)

sinh

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the hyperbolic sine of the argument x.

Usage

```
double sinh (double x)
```

Arguments

- x

The argument with which to calculate the hyperbolic sine.

Return Value

- double

Returns the hyperbolic sine of x.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 3.14159;
...
result = sinh(target);
...
```

Related Topics

[asin](#)

[<math.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[Nucleus POSIX Run Time Library APIs](#)

snprintf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function places an output followed by the null byte, ‘\0’, in consecutive bytes starting at *s. Additionally, the n argument states the size of the buffer referred to by s.

Usage

```
int snprintf (char      *s,  
             size_t    n,  
             const char *format, ...)
```

Arguments

- s
Pointer to the buffer.
- n
Specifies the size of the buffer referred to by s.
- format
Formatting string.

Supported Flag Characters

- -
The result of the conversion is left justified within the field.
- 0
For d, i, o, u, x, X, , e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width.

Supported Length Modifiers

- l
Specifies that a following d,i,o,u, x,orX conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- d,i
The int argument is converted to a signed decimal in the style “[]dddd”.
- o
The unsigned argument is converted to unsigned octal format in the style “dddd”.
- u
The unsigned argument is converted to unsigned decimal format in the style “dddd”

- **x**
The unsigned argument is converted to unsigned hexadecimal format in the style “dddd”
- **X**
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.
- **f,F**
The double argument is converted to decimal notation in the style “[ddd.ddd]”.
- **e,E**
The double argument is converted in the style “[d.ddde±dd]”.
- **g,G**
The double argument is converted in the style f or e (or in the style F or E in the case of a G conversion specifier).
- **c**
The int argument is converted to an unsigned char, and the resulting byte will be written.
- **s**
The argument is a pointer to an array of char. Bytes from the array are written up to (but not including) any terminating null byte.
- **%**
Print a ‘%’ character; no argument is converted. The complete conversion specification will be%%.

Return Value

- **int**
Upon successful completion, the function returns the number of bytes that would have been written to s had n been sufficiently large excluding the terminating null byte.
If the value of n is zero on a call to snprintf(), nothing is written. The number of bytes that would have been written had n been sufficiently large excluding the terminating null will be returned, and s may be a null pointer.

- **Negative Value**

In case of failure it returns a negative value and errno is set to the following error code:

E_OVERFLOW

The file is a regular file and the size of the file cannot be represented correctly in an object of type off_t.

Example

```
#include "services/stdio.h"
```

```
int j;  
char s[] = "Computer";  
char buffer[200];  
  
j = snprintf(buffer,200,"\tString:%s\n",s);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [cosh](#)

<math.h> [tanh](#)

[Nucleus POSIX Run Time Library APIs](#)

sprintf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function places an output followed by the null byte, ‘\0’, in consecutive bytes starting at *s.

Usage

```
int sprintf (char *s,  
            const char *restrict format, ...)
```

Arguments

- s
Pointer to the buffer.
- format
Formatting string.

Supported Flag Characters

- -
The result of the conversion will be left justified within the field.
- 0
For d, i, o, u, x, X, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width.

Supported Length Modifiers

- l
Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- d,i
The int argument is converted to a signed decimal in the style “[d]ddd”.
- o
The unsigned argument is converted to unsigned octal format in the style “ddd”.
- u
The unsigned argument is converted to unsigned decimal format in the style “ddd”.
- x
The unsigned argument is converted to unsigned hexadecimal format in the style “ddd”.
- X
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.

- **f,F**
The double argument is converted to decimal notation in the style “[ddd.ddd]”.
- **e,E**
The double argument is converted in the style “[d.ddde±dd]”.
- **g,G**
The double argument is converted in the style f or e (or in the style F or E in the case of a G conversion specifier)
- **c**
The int argument is converted to an unsigned char, and the resulting byte is written.
- **s**
The argument is a pointer to an array of char. Bytes from the array are written up to (but not including) any terminating null byte.
- **%**
Print a ‘%’ character; no argument is converted. The complete conversion specification is%%.

Return Value

- **int**
On successful completion, returns the number of bytes written.
- **Negative Value**
In case of failure it returns a negative value.

Example

```
#include "services/stdio.h"

int j;
char s[] = "Computer";
char buffer[200];

j = sprintf(buffer, "\tString:%s\n", s);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fputc](#)

[fscanf](#)

[<stdio.h>](#)

[<wchar.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

sqrt

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the square root of the argument x.

Usage

```
double sqrt (double x)
```

Arguments

- x
The argument with which to take the square root.

Return Value

- double
Returns the square root of x.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 25;
...
result = sqrt(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<math.h>](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

srand

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to [rand](#).

Usage

```
void srand (unsigned seed)
```

Arguments

- seed
Seed value.

Example

```
#include "services/stdlib.h"

static int result;
...
srand(10);
result = rand();
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<math.h>](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

sscanf

Allowed from: POSIX Thread and signal handler

This function reads from the string s.

Usage

```
int sscanf (const char *s,  
            const char *format,  
            ... )
```

Arguments

- s
Pointer to the buffer.
- format
Formatting string.

Supported Length Modifiers

- l
Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or unsigned long argument.
- h
Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short.

Supported Conversion Specifiers

- d
Matches an optionally signed decimal integer.
- i
Matches an optionally signed integer.
- o
Matches an optionally signed octal integer.
- u
Matches an optionally signed decimal integer.
- x
Matches an optionally signed hexadecimal integer.
- a,e,f,g
Matches an optionally signed floating-point number.

- **s**
Matches a sequence of bytes that are not white-space characters
- **[**
Matches a non-empty sequence of bytes from a set of expected bytes (the scanset).
- **c**
Matches a sequence of bytes of the number specified by the field width.
- **%**
Matches a single ‘%’ character; no conversion or assignment occurs. The complete conversion specification is%%.

Return Value

- **int**
Number of successfully matched and assigned input items.
- **EOF**
If error encountered.

Example

```
#include "services/stdio.h"

char tokenstring[] = "15 12 14...";

/* Input various data from tokenstring */
sscanf(tokenstring, "%s", s);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getc](#)

[printf](#) [strtod](#)

[strtoul](#) [<stdio.h>](#)

[<wchar.h>](#) [Nucleus POSIX Run Time Library APIs](#)

strcat

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function appends a copy of the string pointed to by s2 (including the terminating null byte) to the end of the string pointed to by s1. The initial byte of s2 overwrites the null byte at the end of s1. If copying takes place between objects that overlap, the behavior is undefined.

Usage

```
char *strcat (char      *s1,  
              const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- char*
Pointer to the value of the destination.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6], *result;  
...  
  
strcpy(string1, "-----");  
strcpy(string2, "abAde");  
...  
  
result = (char *) strcat(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strncat](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strchr

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function locates the first occurrence of *c* (converted to a char) in the string pointed to by *s*. The terminating null byte is considered to be part of the string.

Usage

```
char *strchr (const char *s,  
              int        c)
```

Arguments

- *s*
Pointer to the target string.
- *c*
Character to look for.

Return Value

- *char **
Pointer to the first occurrence of *c*.
- *NULL*
If *c* is not found in *s*

Example

```
#include "services/string.h"  
  
...  
static char string1[6], *result;  
...  
  
strcpy(string1, "abAde");  
...  
result = (char *) strchr(string1, 'A');  
...  

```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strchr](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strcmp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function compares the string pointed to by s1 to the string pointed to by s2.

Usage

```
int strcmp (const char *s1,  
            const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- > 0
s1 is greater than s2.
- 0
s1 is equal to s2.
- < 0
s1 is less than s2.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6];  
static int flag;  
...  
  
strcpy(string1, "-----");  
strcpy(string2, "abAde");  
...  
  
flag = strcmp(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strncmp](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strcoll

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function compares the string pointed to by s1 to the string pointed to by s2, both are interpreted as appropriate to the LC_COLLATE category of the current locale.

Usage

```
int strcoll (const char *s1,
            const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- > 0
s1 is greater than s2.
- 0
s1 is equal to s2.
- < 0
s1 is less than s2.

Example

```
#include "services/string.h"

...
struct node { /* These are stored in the table. */
    char *string;
    int length;
};
...

int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
                  ((const struct node *)node2)->string);
}
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strcmp](#)
[strxfrm](#) [<string.h>](#)

Nucleus POSIX Run Time Library APIs

strcpy

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copies the string pointed to by s2 (including the terminating null byte) into the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.

Usage

```
char *strcpy (char *s1,  
              const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- char*
Pointer to the first string.

Example

```
#include "services/string.h"  
  
...  
static char permstring[11];  
...  
  
strcpy(permstring, "-----");  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strncat](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strcspn

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the length (in bytes) of the maximum initial segment of the string pointed to by s1, which consists entirely of bytes not from the string pointed to by s2.

Usage

```
size_t strcspn (const char *s1,  
               const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- size_t
Length of s1.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6];  
static size_t length;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "--A--");  
  
...  
length = strcspn(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strspn](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strdup

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function returns a pointer to a new string, which is a duplicate of the string pointed to by s1. The returned pointer can be passed to the [free](#) function. A null pointer is returned if the new string cannot be created.

Usage

```
char *strdup (const char *s1)
```

Arguments

- s1
Target string.

Return Value

- char *
Pointer to a new string.
- NULL
If error is encountered.

errno Values

- ENOMEM
Insufficient storage space is available.

Example

```
#include "services/string.h"

...
static char string1[6], string2[6];
static size_t length;
...

strcpy(string1, "abAde");
...

string2 = strdup(string1);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [free](#)
[malloc](#) [<string.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

strerror

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function maps the error number in `errnum` to a locale-dependent error message string and returns a pointer to it. Typically, the values for `errnum` come from `errno`, but this function maps any value of type `int` to a message. This function is non-reentrant. Refer to [strerror_r](#) for the reentrant version of this function.

Usage

```
char *strerror (int errnum)
```

Arguments

- `errnum`
Error number.

Return Value

- `char *`
Pointer to the generated message string.
- `EINVAL`
The value of `errnum` is not a valid error number.

Example

```
#include "services/string.h"

...
static char string1[6];
...

string1 = strerror(255);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [perror](#)

[<string.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

strerror_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function maps the error number in `errnum` to a locale- dependent error message string and returns it in the buffer provided by you. Typically, the values for `errnum` come from `errno`, but this function maps any value of type `int` to a message. This is the re-entrant version of [strerror](#).

Usage

```
int strerror_r (int    errnum,
               char    *strerrbuf,
               size_t  buflen)
```

Arguments

- `errnum`
Error number.
- `strerrbuf`
Buffer to be filled with the error string.
- `buflen`
Length of the buffer.

Return Value

- 0
Indicates successful completion of this function.
- `ERANGE`
In case if insufficient buffer length was provided.

Example

```
#include "services/string.h"

int  status;
char strerrbuf[30];

status = strerror_r(errnum, (char*)strerrbuf, sizeof(strerrbuf));
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [perror](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strftime

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

The function places bytes into the array pointed to by *s* as controlled by the string pointed to by *format*.

Usage

```
size_t strftime (char          *s,  
                 size_t        maxsize,  
                 const char*    format,  
                 const struct tm *timeptr)
```

Arguments

- *s*
Pointer to the string.
- *maxsize*
Maximum size of the number of bytes to be placed in the character array.
- *format*
Given format
- *timeptr*
Pointer to the *tm* structure.

Supported Conversion Specifiers

- *a*
Replaced by the locale's abbreviated weekday name. [*tm_wday*]
- *A*
Replaced by the locale's full weekday name. [*tm_wday*]
- *b*
Replaced by the locale's abbreviated month name. [*tm_mon*]
- *B*
Replaced by the locale's full month name. [*tm_mon*]
- *c*
Replaced by the locale's appropriate date and time representation.
- *C*
Replaced by the year divided by 100 and truncated to an integer, as a decimal number [00,99]. [*tm_year*]
- *d*
Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]

- **D**
Equivalent to %m/%d/%y. [tm_mon, tm_mday, tm_year]
- **e**
Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [tm_mday]
- **F**
Equivalent to %Y-%m-%d (the ISO 8601: 2000 standard date format). [tm_year, tm_mon, tm_mday]
- **g**
Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [tm_year, tm_wday, tm_yday]
- **G**
Replaced by the week-based year (see below) as a decimal number (for example, 1977). [tm_year, tm_wday, tm_yday]
- **h**
Equivalent to %b. [tm_mon]
- **H**
Replaced by the hour (24-hour clock) as a decimal number [00,23]. [tm_hour]
- **I**
Replaced by the hour (12-hour clock) as a decimal number [01,12]. [tm_hour]
- **j**
Replaced by the day of the year as a decimal number [001,366]. [tm_yday]
- **m**
Replaced by the month as a decimal number [01,12]. [tm_mon]
- **M**
Replaced by the minute as a decimal number [00,59]. [tm_min]
- **n**
Replaced by a <newline>.
- **p**
Replaced by the locale's equivalent of either a.m. or p.m. [tm_hour]
- **r**
Replaced by the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S%p. [tm_hour, tm_min, tm_sec]

- **R**
Replaced by the time in 24-hour notation (%H:%M). [tm_hour, tm_min]
- **S**
Replaced by the second as a decimal number [00,60]. [tm_sec]
- **t**
Replaced by a <tab>.
- **T**
Replaced by the time (%H:%M:%S). [tm_hour, tm_min, tm_sec]
- **u**
Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [tm_wday]
- **U**
Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [tm_year, tm_wday, tm_yday]
- **V**
Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [tm_year, tm_wday, tm_yday]
- **w**
Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [tm_wday]
- **W**
Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [tm_year, tm_wday, tm_yday]
- **x**
Replaced by the locale's appropriate date representation.
- **X**
Replaced by the locale's appropriate time representation.
- **y**
Replaced by the last two digits of the year as a decimal number [00,99]. [tm_year]

- **Y**
Replaced by the year as a decimal number (for example, 1997). [tm_year]
- **z**
Replaced by the offset from UTC in the ISO 8601: 2000 standard format (+hhmm or .hhmm), or by no characters if no time zone is determinable. For example, ".0430" CX means 4 hours 30 minutes behind UTC (west of Greenwich). If tm_isdst is zero, the standard time offset is used. If tm_isdst is greater than zero, the daylight savings time offset is used. If tm_isdst is negative, no characters are returned. [tm_isdst]
- **Z**
Replaced by the time zone name or abbreviation or by no bytes if no time zone information exists. [tm_isdst]
- **%**
Replaced by %.

Example

```
#include "services/time.h"

char    string[256];
struct tm time_ptr;
size_t no_bytes_placed;

time_ptr.tm_year  = 2001 - 1900;
time_ptr.tm_month = 7 - 1;
time_ptr.tm_mday  = 4;
time_ptr.tm_hour  = 0;
time_ptr.tm_min   = 0;
time_ptr.tm_sec   = 1;
time_ptr.tm_isdst = -1;

/* Convert the broken down time in to %m/%d/%y format */
no_bytes_placed = strftime(string, sizeof(string), "%D", &time_ptr);
```

Related Topics

asctime	Base Definitions volume of IEEE Std 1003.1-2001
clock	ctime
difftime	getdate
gmtime	LC_TIME
localtime	mktime
time	<time.h>
tzset	utime

[Nucleus POSIX Run Time Library APIs](#)

strlen

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the number of bytes in the string *s* points to, not including the terminating null byte.

Usage

```
size_t strlen (const char *s)
```

Arguments

- *s*
Target string.

Return Value

- *size_t*
Length of *s*.

Example

```
#include "services/string.h"

...
struct element
{
    char *key;
    char *data;
};

...
char *key, *data;
int len;
*keylength = *datalength = 0;
...
if ((len = strlen(key)) > *keylength)
    *keylength = len;
if ((len = strlen(data)) > *datalength)
    *datalength = len;
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strncat

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function appends no more than n bytes (a null byte and bytes that follow it are not appended) from the array pointed to by s2 to the end of the string pointed to by s1.

Usage

```
char *strncat (char      *s1,  
               const char *s2,  
               size_t    n)
```

Arguments

- s1
First string.
- s2
Second string.
- n
Number of bytes to append.

Return Value

- char*
Pointer to the first string.

Description

The initial byte of s2 overwrites the null byte at the end of s1. A terminating null byte is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6], *result;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "--A--");  
...  
  
result = strncat(string1, string2, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[streat](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strncmp

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function compares no more than n bytes (bytes that follow a null byte are not compared) from the array pointed to by s1 to the array pointed to by s2.

Usage

```
int strncmp (const char *s1,
             const char *s2,
             size_t      n)
```

Arguments

- s1
First string.
- s2
Second string.
- n
Number of bytes to compare.

Return Value

- > 0
s1 is greater than s2.
- 0
s1 is equal to s2.
- < 0
s1 is less than s2.

Example

```
#include "services/string.h"

...
static char string1[6], string2[6];
static int flag;
...

strcpy(string1, "abAde");
strcpy(string2, "--A--");
...

flag = strncmp(string1, string2, 4);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strcmp](#)

<string.h>

Nucleus POSIX Run Time Library APIs

strncpy

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copies no more than n bytes (bytes that follow a null byte are not copied) from the array pointed to by s2 to the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.

Usage

```
char *strncpy (char      *s1,  
               const char *s2,  
               size_t    n)
```

Arguments

- s1
First string.
- s2
Second string.
- n
Number of bytes to copy.

Return Value

- s1
First string.

Example

```
#include "services/string.h"  
  
...  
static char string1[6];  
...  
  
strncpy(string1, "abAde", 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strcpy](#)

<string.h> [Nucleus POSIX Run Time Library APIs](#)

strpbrk

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function locates the first occurrence in the string pointed to by s1 of any byte from the string pointed to by s2.

Usage

```
char *strpbrk (const char *s1,  
               const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- char *
Pointer to the byte.
- NULL
No byte from s2 occurs in s1.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6], *result;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "--A--");  
...  
  
result = strpbrk(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strchr](#)
[strchr](#) [<string.h>](#)
[Nucleus POSIX Run Time Library APIs](#)

strchr

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function locates the last occurrence of *c* (converted to a char) in the string pointed to by *s*. The terminating null byte is considered to be part of the string.

Usage

```
char *strchr (const char *s,  
              int        c)
```

Arguments

- *s*
Target string.
- *c*
Character to look for.

Return Value

- *char **
Pointer to the byte.
- *NULL*
c does not occur in *s*.

Example

```
#include "services/string.h"  
  
...  
const char *name;  
char *basename;  
...  
  
basename = strchr(name, '/') + 1;  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strchr](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strspn

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function computes the length (in bytes) of the maximum initial segment of the string pointed to by s1, which consists entirely of bytes from the string pointed to by s2.

Usage

```
size_t strspn (const char *s1,  
               const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- size_t
Length of s1.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6];  
static size_t length;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "abA");  
...  
  
length = strspn(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strcspn](#)

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strstr

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function returns a pointer to the first occurrence in string s1 of the string s2 (not including the terminating null character).

Usage

```
char *strstr (const char *s1,  
              const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- s1
s2 points to a zero-length string.
- 0
s2 does not exist in s1.
- char *
Pointer to the located string.
- NULL
String is not found.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6], *result;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "abA");  
  
result = strstr(string1, string2);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [strchr](#)

[<string.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

strtod

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by `nptr` to a double.

Usage

```
double strtod (const char *nptr,  
               char      **endptr)
```

Arguments

- `nptr`
Pointer to the string converted to double.
- `endptr`
Pointer to the final string stored in the object pointed to by `endptr` provided that `endptr` is not a null pointer.

Return Value

- `double`
Successful completion converted value is returned.
- `NULL`
No conversion can be performed. `errno` is set to the corresponding error.

Example

```
#include "services/string.h"  
#include "services/stdlib.h"  
  
static char string[7], **endptr;  
static double result;  
...  
strcpy(string, "abcdef");  
...  
result = strtod(string, endptr); /* endptr should not be null */  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	<float.h>
isspace	scanf
strtol	<stdlib.h>
Nucleus POSIX Run Time Library APIs	

strtof

This function converts a string to a (floating-point) double-precision number.

Usage

```
float strtof(const char *restrict nptr,  
            char **restrict endptr)
```

Arguments

- `nptr`
Pointer to the number string.
- `endptr`
Pointer to the final string.

Return Values

- `float`
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:

`ERANGE`

The value to be returned is not representable.

`EINVAL`

The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strtoumax

This function converts a string to an integer type.

Usage

```
intmax_t strtoumax(const char *restrict nptr,  
                  char **restrict endptr,  
                  int base)
```

Arguments

- `nptr`
Pointer to the number string.
- `endptr`
Pointer to the final string.
- `base`
Base conversion value.

Return Values

- `intmax_t`
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:
 - ERANGE
The value to be returned is not representable.
 - EINVAL
The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strtok

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a byte from the string pointed to by s2. This function is non-reentrant. Refer to [strtok_r](#) for the reentrant version of this function.

Usage

```
char *strtok (char *s1,  
              const char *s2)
```

Arguments

- s1
First string.
- s2
Second string.

Return Value

- char *
Pointer to the first byte of a token.
- NULL
There is no token.

Example

```
#include "services/string.h"  
  
...  
char *token;  
char *line = "LINE TO BE SEPARATED";  
char *search = " ";  
  
/* Token will point to "LINE". */  
token = strtok(line, search);  
  
/* Token will point to "TO". */  
token = strtok(NULL, search);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <string.h>

[Nucleus POSIX Run Time Library APIs](#)

strtok_r

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a byte from the string pointed to by s2. This is the re-entrant version of [strtok](#).

Usage

```
char *strtok_r (char      *s1,  
                const char *s2,  
                char      **lasts)
```

Arguments

- s1
First string.
- s2
Second string.
- lasts
To be filled with the information necessary to continue scanning the same string.

Return Value

- char *
Pointer to the first byte of a token.
- NULL
There is no token.

Example

```
#include "services/string.h"  
  
...  
char *token;  
char *line = "LINE TO BE SEPARATED";  
char *search = " ";  
char *lasts;  
  
/* Token will point to "LINE". */  
token = strtok_r(line, search,&lasts);  
  
/* Token will point to "TO". */  
token = strtok_r(NULL, search,&lasts);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<string.h>

[Nucleus POSIX Run Time Library APIs](#)

strtol

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by str to a long.

Usage

```
long strtol (const char *str,  
             char      **endptr,  
             int        base)
```

Arguments

- str
Pointer to the string converted to long.
- endptr
Pointer to the final string stored in the object pointed to by endptr provided that endptr is not a null pointer.
- base
Base used for the conversion.

Return Value

- long
Upon successful completion converted value is returned.
- NULL
No conversion can be performed. errno is set to the corresponding error.

Example

```
#include "services/string.h"  
#include "services/stdlib.h"  
  
static char string[7], **endptr;  
static double result;  
...  
strcpy(string, "abcdef");  
...  
result = strtol(string, endptr, 10); /* endptr should not be null */  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [isalpha](#)

[scanf](#)

[strtod](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

strtoll

This function converts a string to a long long number.

Usage

```
long long strtoll(const char *restrict nptr,  
                  char **restrict endptr,  
                  int base)
```

Arguments

- **nptr**
Pointer to the number string.
- **endptr**
Pointer to the final string.
- **base**
Determines the radix of the integer representation.

Return Values

- **long long**
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:

ERANGE

The value to be returned is not representable.

EINVAL

The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strtold

This function converts a string to a (long double) double-precision number.

Usage

```
long double strtold(const char *restrict nptr,  
                   char **restrict endptr)
```

Arguments

- `nptr`
Pointer to the number string.
- `endptr`
Pointer to the final string.

Return Values

- `long double`
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:

`ERANGE`

The value to be returned is not representable.

`EINVAL`

The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strtoul

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts the initial portion of the string pointed to by str to an unsigned long.

Usage

```
long strtoul (const char *str,  
              char      **endptr,  
              int        base)
```

Arguments

- str
Pointer to the string converted to unsigned long.
- endptr
Pointer to the final string stored in the object pointed to by endptr provided that endptr is not a null pointer.
- base
Base used for the conversion.

Return Value

- unsigned long
Upon successful completion converted value is returned.
- NULL
No conversion can be performed. errno is set to the corresponding error.

Example

```
#include "services/string.h"  
#include "services/stdlib.h"  
  
static char string[7], **endptr;  
static double result;  
...  
strcpy(string, "abcdef");  
...  
result = strtoul(string, endptr, 10);/* endptr should not be null */  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[isalpha](#)

[scanf](#)

[strtod](#)

[strtol](#)

[<stdlib.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

strtoull

This function converts a string to an unsigned long long number.

Usage

```
unsigned long long strtoull(const char *restrict str,  
                           char **restrict endptr,  
                           int base)
```

Arguments

- **str**
Pointer to the number string.
- **endptr**
Pointer to the final string.
- **base**
Determines the radix of the integer representation.

Return Values

- **unsigned long long**
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:

ERANGE

The value to be returned is not representable.

EINVAL

The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strtoumax

This function converts a string to an unsigned integer type.

Usage

```
uintmax_t strtoumax(const char *restrict nptr,  
                    char **restrict endptr,  
                    int base)
```

Arguments

- **nptr**
Pointer to the number string.
- **endptr**
Pointer to the final string.
- **base**
Determines the radix of the integer representation.

Return Values

- **uintmax_t**
The converted value (if any), zero if no conversion could be performed, or the corresponding error code:
 - ERANGE**
The value to be returned is not representable.
 - EINVAL**
The value of base is not supported.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX Run Time Library APIs](#)

strxfrm

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function transforms the string pointed to by s2 and places the resulting string into the array pointed to by s1.

Usage

```
size_t strxfrm (char      *s1,  
                const char *s2,  
                size_t    n)
```

Arguments

- s1
First string.
- s2
Second string.
- n
Number of bytes to place in the resulting array.

Return Value

- size_t
Length of the transformed string (not including the terminating null byte).

Description

The transformation is such that if [strcmp](#) is applied to two transformed strings, it returns a value greater than, equal to, or less than 0, corresponding to the result of [strcmp](#) applied to the same two original strings. No more than n bytes are placed into the resulting array pointed to by s1, including the terminating null byte. If n is 0, s1 is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

Example

```
#include "services/string.h"  
  
...  
static char string1[6], string2[6];  
static size_t length;  
...  
  
strcpy(string1, "abAde");  
strcpy(string2, "abA");  
...  
  
length = strxfrm(string1, string2, 4);  
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[strcoll](#)

[Nucleus POSIX Run Time Library APIs](#)

[strcmp](#)

[<string.h>](#)

swab

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function copies nbytes bytes, which are pointed to by src, to the object pointed to by dest, exchanging adjacent bytes.

Usage

```
void swab (const void *src,  
           void      *dest,  
           ssize_t   nbytes)
```

Arguments

- src
Pointer to the source.
- dest
Pointer to the destination.
- nbytes
Number of bytes to copy.

Example

```
#include "services/unistd.h"  
  
char dest[6];  
  
/* Copies four bytes from source to destination. */  
swab((void*)"test", (void*)dest, 4);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<unistd.h>

[Nucleus POSIX Run Time Library APIs](#)

sysconf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function provides a method for the application to determine the current value of a configurable system limit or option (variable).

Usage

```
long sysconf (int name)
```

Arguments

- name
System variable to be queried.

Return Value

- long
Current value of a configurable system limit, on success.
- POSIX_ERROR
Error encountered. Sets errno to the following error code:
EINVAL
The value of the name argument is not valid.

Example

```
#include "services/unistd.h"

long value;

/* Find the System wide Maximum path length limit */
value = sysconf(PATH_MAX);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	confstr
fgetcon	<limits.h>
pathconf	Shell and Utilities volume of IEEE Std 1003.1-2001
<unistd.h>	Nucleus POSIX Run Time Library APIs

tan

This function computes the compute the tangent of their argument x, measured in radians.

Usage

```
double tan (double x)
```

Arguments

- x
The argument with which to calculate the tangent.

Return Value

- double
Returns the tangent of x.

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 0.778;
...
result = tan(target);
...
```

Related Topics

[atan](#)

[<math.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[Nucleus POSIX Run Time Library APIs](#)

tanh

This function computes the hyperbolic tangent of their argument x , measured in radians.

Usage

```
double tanh (double x)
```

Arguments

- x
The argument with which to calculate the hyperbolic tangent.

Return Value

- `double`
Returns the hyperbolic tangent of x .

Example

```
#include "services/math.h"

static double target;
static double result;
...
target = 0.778;
...
result = tanh(target);
...
```

Related Topics

[atanh](#)

Base Definitions volume of IEEE Std 1003.1-2001

[tan](#)

[<math.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

time

Allowed from: POSIX Thread and signal handler

This function returns the value of time in seconds since the epoch.

Usage

```
time_t time (time_t *tloc)
```

Arguments

- `tloc`
Pointer to an area where the return value is also stored.

Return Value

- `time_t`
On successful completion, returns the value of time.
- `(time_t)-1`
An error is encountered.

Example

```
#include "services/time.h"

time_t time_value;

/* Find the current value of time since Epoch */
time_value = time(NULL);
```

Related Topics

asctime	Base Definitions volume of IEEE Std 1003.1-2001
clock	ctime
difftime	gmtime
localtime	strftime
<time.h>	tzset
utime	Nucleus POSIX Run Time Library APIs

tmpfile

Allowed from: POSIX Thread and signal handler

This function creates a temporary file and opens a corresponding stream.

Usage

```
FILE *tmpfile (void)
```

Return Value

- **FILE ***

Pointer to the stream of the file that is created.

- **NULL**

If an error is encountered, this function returns a null pointer and sets `errno` to the corresponding error code:

EACCES

Permission denied.

EMFILE

{OPEN_MAX} file descriptors are currently open in the calling process.

ENAMETOOLONG

The length of the filename argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

ENOENT

A component of filename does not name an existing file or filename is an empty string.

ENOSPC

The directory or file system that will contain the new file cannot be expanded, the file does not exist, and the file was to be created.

EINVAL

The value of the mode argument is not valid.

ENAMETOOLONG

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

ENOMEM

Insufficient storage space is available.

EEXIST

The named file exists.

EIO

An I/O error occurred while accessing the file system.

Example

```
#include "services/stdio.h"

...
FILE *fp;

fp = tmpfile();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[tmpnam](#)

[unlink](#)

[fopen](#)

[<stdio.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

tmpnam

Allowed from: POSIX Thread and signal handler

This function generates a string that is a valid filename and that is not the same as the name of an existing file. This function is non-reentrant.

Usage

```
char *tmpnam (char *s)
```

Arguments

- `s`
Pointer to the string to be filled with the filename.

Return Value

- `char *`
On successful completion, returns the pointer to the string containing file name.
- `NULL`
If an error is encountered, this function returns a null pointer and sets `errno` to the corresponding error code:

`EACCES`

Permission denied.

`EMFILE`

{`OPEN_MAX`} file descriptors are currently open in the calling process.

`ENAMETOOLONG`

The length of the filename argument exceeds {`PATH_MAX`} or a pathname component is longer than {`NAME_MAX`}.

`ENOENT`

A component of filename does not name an existing file or filename is an empty string.

`ENOSPC`

The directory or file system that will contain the new file cannot be expanded, the file does not exist, and the file was to be created.

`EINVAL`

The value of the mode argument is not valid.

`ENAMETOOLONG`

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {`PATH_MAX`}.

`ENOMEM`

Insufficient storage space is available.

EEXIST

The named file exists.

EIO

An I/O error occurred while accessing the file system.

Example

```
#include "services/stdio.h"

...

char filename[L_tmpnam+1];
char *ptr;
ptr = tmpnam(filename);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	fopen
open	tmpfile
unlink	<stdio.h>
Nucleus POSIX Run Time Library APIs	

toascii

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts its argument into a 7-bit ASCII character.

Usage

```
int toascii (int c)
```

Arguments

- `c`
Character converted to ASCII.

Return Value

- `int`
Character converted to ASCII.

Example

```
#include "services/ctype.h"

static int target, result;
...
target = 65;
...
result = toascii(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [isascii](#)

[<ctype.h>](#)

[Nucleus POSIX Run Time Library APIs](#)

tolower

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts its argument into a lowercase character.

Usage

```
int tolower (int c)
```

Arguments

- `c`
Character to be converted to lowercase.

Return Value

- `int`
Character converted to lowercase.

Example

```
#include "services/ctype.h"

static int target, result;
...
target = 65;
...
result = tolower(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 `<ctype.h>`

[Nucleus POSIX Run Time Library APIs](#)

toupper

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function converts its argument into an uppercase character.

Usage

```
int toupper (int c)
```

Arguments

- `c`
Character to be converted to uppercase.

Return Value

- `int`
Character converted to uppercase.

Example

```
#include "services/ctype.h"

static int target, result;
...
target = 65;
...
result = toupper(target);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 `<ctype.h>`

[Nucleus POSIX Run Time Library APIs](#)

ungetc

Allowed from: POSIX Thread and signal handler

This function pushes the byte specified by *c* (converted to an unsigned char) back onto the input stream pointed to by *stream*.

Usage

```
int ungetc (int c,  
            FILE *stream)
```

Arguments

- *c*
Specifies the byte to be pushed back.
- *stream*
Pointer to a file stream.

Return Value

- *int*
Byte pushed back after conversion.
- EOF
If an error is encountered, *errno* is to the corresponding error code:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"  
#include "services/ctype.h"  
  
int ch;  
int result = 0;  
  
/* Reading and convert number */  
while ( ((ch = getchar()) != EOF) && isdigit(ch) )
```

```
    result = result * 10 + ch - '0';  
  
    if (ch != EOF)  
        ungetc(ch,stdin);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fsetpos](#)

[read](#)

[<stdio.h>](#)

[fseek](#)

[getc](#)

[rewind](#)

[Nucleus POSIX Run Time Library APIs](#)

unsetenv

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function removes an environment variable from the environment of the calling process.

Usage

```
int unsetenv (const char *name)
```

Arguments

- **name**
Pointer to the name of the environment variable to be removed.

Return Value

- **POSIX_SUCCESS**
Successful completion.
- **POSIX_ERROR**
Indicates failure of the service and sets the errno to the corresponding error:
EINVAL
The name argument is a null pointer, points to an empty string, or points to a string containing an '=' character.

Example

```
#include "services/stdlib.h"

...
unsetenv("PATH");
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[setenv](#)

<sys/types.h>

[Nucleus POSIX Run Time Library APIs](#)

[getenv](#)

<stdlib.h>

<unistd.h>

fprintf

Allowed from: POSIX Thread and signal handler

This function places output on the named output stream. Instead of being called with a variable number of arguments (as in [fprintf](#)), the `fprintf` function is called with an argument list as defined by `<stdarg.h>`.

Usage

```
int fprintf (FILE      *stream,  
            const char *format,  
            va_list    ap)
```

Arguments

- `stream`
Pointer to a file stream.
- `format`
Output formatting string.
- `ap`
List of arguments.

Supported Flag Characters

- `-`
The result of the conversion will be left justified within the field.
- `0`
For `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width;

Supported Length Modifiers

- `l`
Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- `d,i`
The `int` argument is converted to a signed decimal in the style “`[]dddd`”.
- `o`
The unsigned argument is converted to unsigned octal format in the style “`dddd`”.
- `u`
The unsigned argument is converted to unsigned decimal format in the style “`dddd`”.

- **x**
The unsigned argument is converted to unsigned hexadecimal format in the style “dddd”.
- **X**
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.
- **f,F**
The double argument is converted to decimal notation in the style “[ddd.ddd”.
- **e,E**
The double argument is converted in the style “[d.ddde±dd”.
- **g,G**
The double argument is converted in the style f or e (or in the style F or E in the case of a G conversion specifier).
- **c**
The int argument is converted to an unsigned char, and the resulting byte will be written.
- **s**
The argument is a pointer to an array of char. Bytes from the array is written up to (but not including) any terminating null byte.
- **%**
Print a ‘%’ character; no argument is converted. The complete conversion specification will be % %.

Return Value

- **int**
Number of bytes on successful completion.
- **Negative Number**
In case of failure, it returns a negative number and sets errno to the corresponding error:

EBADF

The file descriptor-underlying stream is not valid.

EIO

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

```
#include "services/stdio.h"

FILE *stream;
char s[] = "This is a string";
char c = '\n';

stream = fopen("stream1", "w");

fprintf(stream, "%s%c", s, c);

fclose(stream);
```

See Also

Note



In all the string and ctype related functions given in the following, in which there is a support of Locale, only “C” locale is supported.

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

<stdarg.h>

[Nucleus POSIX Run Time Library APIs](#)

[fprintf](#)

<stdio.h>

vsnprintf

Allowed from: Application Initialize, LISR, HISR, POSIX Thread and signal handler

This function places an output followed by the null byte, ‘\0’, in consecutive bytes starting at *s. Instead of being called with a variable number of arguments, the vsnprintf function is called with an argument list as defined by <stdarg.h>.

Usage

```
int vsnprintf (char      *s,  
              size_t    n,  
              const char *format,  
              va_list    ap)
```

Arguments

- s
Pointer to the buffer.
- n
Specifies the size of the buffer referred to by s.
- format
Formatting string.
- ap
Argument list.

Supported Flag Characters

- -
The result of the conversion are left justified within the field.
- 0
For d, i, o, u, x, X, , e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width.

Supported Length Modifiers

- l
Specifies that a following d,i,o,u, x,orX conversion specifier applies to a long or unsigned long argument.

Supported Conversion Specifiers

- d,i
The int argument is converted to a signed decimal in the style “[]dddd”.
- o
The unsigned argument is converted to unsigned octal format in the style “dddd”.

- **u**
The unsigned argument is converted to unsigned decimal format in the style “dddd”
- **x**
The unsigned argument is converted to unsigned hexadecimal format in the style “dddd”
- **X**
Equivalent to the x conversion specifier, except that letters “ABCDEF” are used instead of “abcdef”.
- **f,F**
The double argument is converted to decimal notation in the style “[]ddd.ddd”.
- **e,E**
The double argument is converted in the style “[]d.ddde±dd”.
- **g,G**
The double argument is converted in the style f or e (or in the style F or E in the case of a G conversion specifier).
- **c**
The int argument is converted to an unsigned char, and the resulting byte will be written.
- **s**
The argument is a pointer to an array of char. Bytes from the array will be written up to (but not including) any terminating null byte.
- **%**
Print a ‘%’ character; no argument is converted. The complete conversion specification will be%%.

Return Value

- **int**
Upon successful completion, the function returns the number of bytes that would have been written to s had n been sufficiently large excluding the terminating null byte.
If the value of n is zero on a call to [snprintf](#), nothing is written. The number of bytes that would have been written had n been sufficiently large excluding the terminating null will be returned, and s may be a null pointer.
- **Negative Value**
In case of failure, it returns a negative value and sets errno to the corresponding error:
EOVERFLOW
The file is a regular file and the size of the file cannot be represented correctly in an object of type off_t.

Example

```
#include "services/stdio.h"

void rtl_vsnprintf()
{
    int    result = 0;
    char   *test_string="vsnprintf";
    double cap = 123.456;

    nprintf_test(result,cap,test_string);
}

void vsnprintf_test(int result,...)
{
    char strnx[200];
    va_list pvar;

    va_start(pvar, result);

    result = vsnprintf(strnx,40,"The number is: %f testing %s \n",pvar);

    va_end(pvar);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[fprintf](#)

<stdarg.h>

<stdio.h>

[Nucleus POSIX Run Time Library APIs](#)

Chapter 4

Nucleus POSIX File System Library APIs

This chapter provides a detailed reference of all the Nucleus POSIX File System Library (POSIX FS) APIs.

- [access](#)
- [chdir](#)
- [close](#)
- [closedir](#)
- [creat](#)
- [dup](#)
- [dup2](#)
- [fcntl](#)
- [fdatasync](#)
- [fseeko](#)
- [fstat](#)
- [fsync](#)
- [ftello](#)
- [ftruncate](#)
- [getcwd](#)
- [I_ALLOCTIMEOUT](#)
- [I_BUFFALLOC](#)
- [I_BUFFDEALLOC](#)
- [I_RTIMEOUT](#)
- [I_SFLAGS](#)
- [I_SIFFPTR](#)
- [I_WTIMEOUT](#)
- [ioctl](#)

- [link](#)
- [lseek](#)
- [mkdir](#)
- [mmap](#)
- [msync](#)
- [munmap](#)
- [open](#)
- [opendir](#)
- [read](#)
- [readdir](#)
- [readdir_r](#)
- [rename](#)
- [rewinddir](#)
- [rmdir](#)
- [seekdir](#)
- [shm_open](#)
- [shm_unlink](#)
- [stat](#)
- [telldir](#)
- [unlink](#)
- [utime](#)
- [write](#)

access

Allowed from: POSIX Thread and signal handler

This function checks a specified file name for accessibility according to the bit pattern given in the amode.

Usage

```
int access (const char *path,  
            int         amode)
```

Arguments

- path
Pointer to path of the file whose accessibility is to be checked.
- amode
Bit pattern of the mode.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:

EINVAL

Invalid value of the bit pattern specified by amode.

ENAMETOOLONG

The length of the path argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

ENOENT

No entry is found corresponding to the given path.

EACCESS

Do not have accessibility permission according to the bit pattern.

Example

```
#include "services/unistd.h"  
  
int ret;  
  
/* Check the accessibility permissions for the file "/home/cnd/mod2" */  
ret = access("/home/cnd/mod2", F_OK);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [stat](#)

<unistd.h

[Nucleus POSIX File System Library APIs](#)

chdir

Allowed from: POSIX Thread and signal handler

This service changes the working directory to the specified path.

Usage

```
int chdir (const char *path)
```

Arguments

- **path**
Pointer to the path of the directory to become the current working directory.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:

ENOTDIR

Not a directory.

EIO

I/O error.

ENAMETOOLONG

Directory name too long.

ENOSPC

No space left on the device.

ENOENT

No such directory.

Example

```
#include "services/unistd.h"

char *directory = "/tmp";
int ret;

ret = chdir (directory);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getcwd](#)

<unistd.h>

[Nucleus POSIX File System Library APIs](#)

close

Allowed from: POSIX Thread and signal handler

This service deallocates the file descriptor indicated by `filedes` (file descriptor).

Usage

```
int close (int filedes)
```

Arguments

- `filedes`
File descriptor to be closed.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EBADF`

Bad file descriptor.

`EACCES`

Permission is denied.

`EIO`

I/O error.

Description

This will make the file descriptor available to any other thread, that is, it releases the thread. All outstanding record locks owned by the thread, on the file associated with the file descriptor are removed (that is, unlocked). It is also possible to close a socket.

This call cannot be interrupted by the signal.

Example

```
#include "services/fcntl.h"

int ret, fd;

fd = open("somefile", O_RDONLY);

/* If no error is encountered*/
ret = close(fd);
```

Related Topics

[accept](#)

[creat](#)

[fopen](#)

[open](#)

[<unistd.h>](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[fclose](#)

[ioctl](#)

[socket](#)

[Nucleus POSIX File System Library APIs](#)

closedir

Allowed from: POSIX Thread and signal handler

This service closes the directory stream referred to by the argument `dirp`.

Usage

```
int closedir (DIR *dirp)
```

Arguments

- `dirp`
Pointer to the directory stream.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EBADF`
Bad file descriptor.

Example

```
#include "services/dirent.h"

...
DIR *dir;
struct dirent *dp;
...

if ((dir = opendir(".")) == NULL)
{
    ...
}

while ((dp = readdir (dir)) != NULL)
{
    ...
}

closedir (dir);
...
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 `<dirent.h>`

[opendir](#)

[Nucleus POSIX File System Library APIs](#)

creat

Allowed from: POSIX Thread and signal handler

This service creates a new file with a specified mode. All files are created with default user mode, which is the current user, and with read-write-execute (S_IRWX) privileges. Nucleus POSIX does not support users or user groups.

Usage

```
int creat (const char *path,  
          mode_t      mode)
```

Arguments

- **path**
Pointer to the path of the file to be created.
- **mode**
Given mode of the file to be created.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and set errno to the corresponding error code:

ENAMETOOLONG

File name is too long.

ENOENT

No such file or directory.

EACCES

Permission is denied.

EMFILE

Maximum number of file descriptors is already open.

ENOSPC

No memory left in the system to allocate to file.

EINVAL

Invalid argument.

EEXIST

The file already exists.

EIO

I/O error.

Example

```
#include "services/fcntl.h"

int fd;

fd = creat("new_file", S_IRWX);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

<fcntl.h> [open](#)

<sys/stat.h> [<sys/types.h>](#)

[Nucleus POSIX File System Library APIs](#)

dup

Allowed from: POSIX Thread and signal handler

This function duplicates an open file descriptor. The dup function provides an alternative interface to the service provided by [fcntl](#) using the F_DUPFD command.

Usage

```
int dup (int fildes)
```

is equivalent to:

```
fid = fcntl (fildes, F_DUPFD, 0)
```

Arguments

- **fildes**
File descriptor associated with the open file, whose information is to be duplicated.

Return Value

- **fd**
Upon successful completion a non-negative integer, namely the file descriptor, will be returned.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:
 - EBADF**
The fildes argument is not a valid open file descriptor.
 - EMFILE**
The number of file descriptors in use by this process would exceed {OPEN_MAX}.

Example

```
#include "services/unistd.h"

int fd;
int fd1;

/* open a file that gives back a file descriptor for
   duplicate operation */

fd = open("tom.txt", O_CREAT | O_RDWR, S_IRWXU);
fd1 = dup(fd);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

[fcntl](#) [open](#)

<unistd.h>

Nucleus POSIX File System Library
APIs

dup2

Allowed from: POSIX Thread and signal handler

This function duplicates an open file descriptor. The dup2 function provides an alternative interface to the service provided by [fcntl](#) using the F_DUPFD command.

Usage

```
int dup2 (int fildes,  
         int fildes2)
```

is equivalent to:

```
close (fildes2);  
fid = fcntl (fildes, F_DUPFD, fildes2);
```

Arguments

- **fildes**
File descriptor associated with the open file, whose information is to be duplicated.
- **fildes2**
The new file descriptor will refer to the same open file description as the original file descriptor.

Return Value

- **fd**
Upon successful completion a non-negative integer, namely the file descriptor, will be returned.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:

EBADF

The fildes argument is not a valid open file descriptor or the argument fildes2 is negative or greater than or equal to {OPEN_MAX}.

EINTR

The dup2 function was interrupted by a signal.

Example

```
#include "services/unistd.h"  
  
int fd,fd1,fd2;  
/* open two files that gives back file descriptors for  
   duplicate operation */  
  
fd  = open("test.txt",O_CREAT | O_RDWR, S_IRWXU);  
fd1 = open("test1.txt",O_CREAT | O_RDWR, S_IRWXU);  
fd2 = dup2(fd,fd1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

[fcntl](#) [open](#)

[<unistd.h>](#) [Nucleus POSIX File System Library APIs](#)

fcntl

Allowed from: POSIX Thread and signal handler

This function performs any of the file operations listed under cmd on the file descriptor fildes.

Usage

```
int fcntl (int fildes,  
           int cmd,  
           ...)
```

Arguments

- fildes
File descriptor.
- cmd

This can be set to any of the following file operations:

F_DUPFD

Duplicate open file.

F_GETFD

Get the file descriptor flags.

F_SETFD

Set the file descriptor flags.

F_GETFL

Get the file status flags and file access modes.

F_SETFL

Set the file status flags.

F_PUSH

This functions sets the push bit for a given output port. (Networking Additional support).

NU_SET_ZC_MODE

There are two arguments values:

NU_ZC_ENABLE

Enable Zero Copy mode on the socket. Note that zero copy is disabled on sockets by default.

NU_ZC_DISABLE

Disable Zero Copy mode on the socket. (Networking Additional support)

NU_SETFLAG

The same option can be done with F_SETFL. There are two arguments values:

NU_BLOCK

blocking on the socket. Note that sockets are blocking by default.

NU_NO_BLOCK

Disable blocking on the socket. (Networking Additional support.)

- **arg** (optional arguments)
File descriptor flags argument of the file.

Return Value

- **F_DUPFD**
A new file descriptor.
- **F_GETFD**
Value of flags defined in `<fcntl.h>`. The return value is not negative.
- **F_SETFD**
Value other than -1.
- **F_GETFL**
Value of file status flags and access modes. The return value is not negative.
- **F_SETFL**
Value other than -1.
- **F_GETLK**
Value other than -1.
- **F_SETLK**
Value other than -1.
- **F_SETLKW**
Value other than -1.
- **F_GETOWN**
Value of the socket owner process or process group; this is not -1.
- **F_SETOWN**
Value other than -1.
- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Sets `errno` to the corresponding error code:

EACCES/ EAGAIN

The `cmd` argument is `F_SETLK`; the type of lock (`l_type`) is a shared (`F_RDLCK`) or exclusive (`F_WRLCK`) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.

EBADF

The `fildest` argument is not a valid open file descriptor, or the argument `cmd` is `F_SETLK` or `F_SETLKW`, the type of lock, `l_type`, is a shared lock (`F_RDLCK`), and `fildest` is not a valid file descriptor open for reading, or the type of lock, `l_type`, is an exclusive lock (`F_WRLCK`), and `fildest` is not a valid file descriptor open for writing.

EINTR

The `cmd` argument is `F_SETLKW` and the function was interrupted by a signal.

EINVAL

The `cmd` argument is invalid, or the `cmd` argument is `F_DUPFD` and `arg` is negative or greater than or equal to `{OPEN_MAX}`, or the `cmd` argument is `F_GETLK`, `F_SETLK`, or `F_SETLKW` and the data pointed to by `arg` is not valid, or `fildest` refers to a file that does not support locking.

EMFILE

The argument `cmd` is `F_DUPFD` and `{OPEN_MAX}` file descriptors are currently open in the calling process, or no file descriptors greater than or equal to `arg` are available.

ENOLCK

The argument `cmd` is `F_SETLK` or `F_SETLKW` and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.

EOVERFLOW

One of the values to be returned cannot be represented correctly.

EOVERFLOW

The `cmd` argument is `F_GETLK`, `F_SETLK`, or `F_SETLKW` and the smallest or, if `l_len` is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type `off_t`.

EDEADLK

The file/channel already exists.

EIO

The `cmd` argument is `F_SETLKW`, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.

Example

```
#include "services/unistd.h"

int fd;
int fd1;

/* open a file that gives back a file descriptor for
   duplicate operation */

fd  = open("tom.txt", O_CREAT | O_RDWR, S_IRWXU);
fd1 = fcntl(fd, F_DUPFD, 0);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

<fcntl.h> [open](#)

[sigaction](#) <signal.h>

<unistd.h> [Nucleus POSIX File System Library APIs](#)

fdatasync

This function forces all currently queued I/O operations associated with the file indicated by file descriptor `fildes` to the synchronized I/O completion state. The functionality is equivalent to [fsync](#).

Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services\unistd"

...
int fdatasync(int fildes)
```

Arguments

- `fildes`
File descriptor

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of this function.
- `POSIX_ERROR`
If the call fails, it returns -1 and sets `errno` to the corresponding error:
 - `EBADF`
The `fildes` argument is not a valid file descriptor open for writing.
 - `EINVAL`
implementation does not support synchronized I/O for this file.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

fseeko

This function sets the file-position indicator for the stream pointed to by stream. It is equivalent to the [fseek](#) function except that the offset argument is of type `off_t`.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "posix_rtl\inc\stdio.h"

...
int fseeko(FILE *stream,
            off_t offset,
            int whence)
```

Arguments

- stream

Pointer to a file stream.

- offset

Offset from the start.

- whence

Determines from where to seek. This can be set to one of the following values:

`SEEK_SET`

The file offset is set to offset bytes.

`SEEK_CUR`

The file offset is set to offset bytes.

`SEEK_END`

The file offset is set to the size of the file plus the offset. Seeking is not allowed beyond the end of the file.

Return Values

- `POSIX_SUCCESS`

Indicates successful completion of this function.

- `POSIX_ERROR`

If the call fails, it returns -1 and sets `errno` to the corresponding error:

`EBADF`

The file descriptor-underlying stream is not valid.

`EIO`

I/O error.

ENOSPC

Insufficient storage space is available.

EACCES

Permission denied.

EINVAL

The stream argument is not a valid stream.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

fstat

Allowed from: POSIX Thread and signal handler

This service obtains information about an open file associated with the file descriptor fields and writes the information to the area pointed to by the buf argument.

Usage

```
int fstat (int      fildes,  
          struct stat *buf)
```

Arguments

- fildes
File descriptor associated with the open file. This service will obtain information from this file.
- buf
Pointer to the buffer to be filled with the file information.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EIO
An error occurred while reading from the file system.
 - EBADF
The fildes argument is not a valid file descriptor.

Example

```
#include "services/sys/types.h"  
#include "services/sys/stat.h"  
#include "services/fcntl.h"  
  
struct stat buffer;  
int fildes;  
int status;  
  
/* Open the file */  
fildes = open("/usr/file1", O_RDWR);  
  
/* Find the information about the above file */  
status = fstat(fildes, &buffer);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [stat](#)

[<sys/stat.h>](#)

[<sys/types.h>](#)

[Nucleus POSIX File System Library APIs](#)

fsync

Allowed from: POSIX Thread and signal handler

This function transfers all the data for the open file descriptor, named by `fildes`, to the storage device associated with the file described by `fildes`.

Usage

```
int fsync(int fildes)
```

Arguments

- `fildes`
File descriptor.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of this function.
- `POSIX_ERROR`
If an error is encountered, `errno` is set to the corresponding error code:

`EBADF`

The `fildes` argument is not a valid descriptor.

`EINVAL`

The `fildes` argument does not refer to a file on which this operation is possible.

Description

This transfer is implementation-defined. The `fsync()` function does not return until the system has completed that action or until an error is detected.

If `_POSIX_SYNCHRONIZED_IO` is defined, the `fsync()` function forces all currently queued I/O operations associated with the file to the synchronized I/O completion state. All I/O operations is completed as defined for synchronized I/O file integrity completion.

Example

```
#include <unistd.h>

int fd;
int status;

fd = open("./test1.txt", O_CREAT | O_RDWR | O_SYNC, S_IRWXU);
if (write(fd, "AA", 2) > 0)
{
    status = fsync(fd);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<unistd.h>](#)

[Nucleus POSIX File System Library APIs](#)

ftello

This function obtains the current value of the file-position indicator for the stream pointed to by stream. It is equivalent to [ftell](#), except that the return value is of type `off_t`.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "posix_rtl\inc\stdio.h"

...
off_t ftello(FILE *stream)
```

Arguments

- `stream`
Pointer to a file stream.

Return Values

- `position`
If the call is successful, it returns the current value of the file-position indicator for the stream, measured in bytes, from the beginning of the file.

- `POSIX_ERROR`
If the call fails, it returns -1 and sets `errno` to the corresponding error:

`EBADF`

The file descriptor-underlying stream is not valid.

`EIO`

I/O error.

`ENOSPC`

Insufficient storage space is available.

`EACCES`

Permission denied.

`EINVAL`

The stream argument is not a valid stream.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

ftruncate

This function truncates the size of a specified file to the specified length.

The Nucleus POSIX product and file system must be successfully initialized to call this function.

Usage

```
#include "services\unistd.h"

...
int ftruncate(int fildes,
              off_t length)
```

Arguments

- **fildes**
The file descriptor to truncate.
- **length**
The length to which to truncate the file size.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of this function.
- **POSIX_ERROR**
If the call fails, it returns -1 and sets **errno** to the corresponding error:
 - EINVAL**
The length argument was less than zero.
 - EBADF**
The fildes argument is not a file descriptor open for writing.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

getcwd

Allowed from: POSIX Thread and signal handler

This service places an absolute pathname of the current working directory in the array pointed to by buf.

Usage

```
char *getcwd (char *buf,  
              size_t size)
```

Arguments

- buf
Pointer to the buffer to be filled with the current directory.
- size
Size of the buffer.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EINVAL
The stream argument is not a valid stream.
 - EACCES
Permission denied.
 - ENOMEM
Not enough memory.
 - ERANGE
Result too large.

Example

```
#include services/stdlib.h  
#include services/unistd.h  
  
...  
  
long size;  
char *buf;  
char *ptr;  
  
size = pathconf(".", _PC_PATH_MAX);
```

```
if ((buf = (char *)malloc((size_t)size)) != NULL)
    ptr = getcwd(buf, (size_t)size);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [malloc](#)

[<unistd.h>](#)

[Nucleus POSIX File System Library APIs](#)

ioctl

Allowed from: POSIX Thread and signal handler

This function can be used to perform a variety of control functions on devices.

Usage

```
int ioctl (int fildes,  
          int request,  
          ...)
```

Arguments

- fildes

Descriptor on which we have to perform I/O control operation.

- request

Selects the control operation to be performed. This can be set to any of the following values:

[I_ALLOCTIMEOUT](#)

Set timeout for the buffer allocation operation.

[I_BUFFALLOC](#)

Allocate buffer for sending data.

[I_BUFFDEALLOC](#)

Deallocates previously allocated receive buffer.

[I_RTIMEOUT](#)

Set timeout for the read from channel operation.

[I_SFLAGS](#)

Set the flags for the corresponding channel.

[I_SIFFPTR](#)

Create a mapping table between the name of interface and pointer to that interface.

[I_WTIMEOUT](#)

Set timeout for the write to channel operation.

- arg (optional arguments)
Any optional arguments.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code.

Description

The request argument and an optional third argument (with varying type) are passed to and interpreted by the appropriate part of the stream associated with the fildes. The fildes argument is an open file descriptor that refers to a device. The request argument selects the control function to be performed and depends upon the device being addressed. The arg argument represents additional information needed by this specific device to perform a requested function. The type of arg depends upon the particular control request, but it is either an integer or a pointer to the device specific data structure. Ioctl does not support any Networking operations.

Example

Note



Refer to [I_ALLOC_TIMEOUT](#), [I_BUFFALLOC](#), [I_BUFFDEALLOC](#), [I_RTIMEOUT](#), [I_SFLAGS](#), [I_SIFFPTR](#) and [I_WTIMEOUT](#) for examples.

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

[open](#)

[read](#)

[sigaction](#)

[<stropts.h>](#)

[write](#)

[Nucleus POSIX File System Library APIs](#)

I_IOCTLTIMEOUT

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) ioctl request arguments. This argument sets the timeout value for the buffer allocation operation.

Usage

```
int ioctl (int          fildes,
           int          request,
           const struct timespec *timeout)
```

Arguments

- **fildes**
Descriptor on which we have to perform I/O control operation.
- **request**
Selects the control operation to be performed.
- **timeout**
Pointer to the structure of type timespec to determine the timeout for the buffer allocation operation. A value of zero means no timeout. This means that buffer allocation will not suspend, but it will return immediately if it is not available.

errno Values

- **EINVAL**
The timeout argument is NULL or specified a nanosecond value less than zero or greater than or equal to 1 billion.

Example

```
#include "services/stropts.h"
#include "services/pipc.h"

int fd;
struct timespec tp;

/* Set the write timeout to 10 ticks. */
tp.tv_sec  = 0;
tp.tv_nsec = 1000000000;

/* Path depends on IPC 1.1x or IPC 2.x, please refer to open() API section
   for details */
fd = open(path, O_NONBLOCK | O_WRONLY);

/* Set the buffer allocation timeout to 10 ticks. */
status = ioctl(fd, I_IOCTLTIMEOUT, &tp);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_BUFFERALLOC

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) ioctl request arguments. This argument allocates the buffer for sending data with the timeout value.

Usage

```
int ioctl (int          fildes,
           int          request,
           unsigned long nbytes,
           void         **buff,
           const char    *ifname,
           const struct timespec *timeout)
```

Arguments

- **fildes**
Descriptor on which to perform I/O control operation.
- **request**
Selects the control operation to be performed.
- **nbytes**
Number of bytes being requested to allocate.
- **buff**
Pointer to the buffer to be filled with the received buffer.
- **ifname**
Name of the physical interface to allocate the buffer for.
- **timeout**
Pointer to the structure of type `timespec` to contain the timeout value.

errno Values

- **EINVAL**
If `buff` argument is `NULL` or `ifname` argument is `NULL` or no pointer to the interface control block is found against the interface name or the `timeout` argument is `NULL` or specified a nanosecond value less than zero or greater than or equal to 1 billion.
- **ENOMEM**
No more memory available to allocate the buffer.

Example

```
#include "services/stropts.h"
#include "services/pipc.h"

int fd;
struct timespec tp;
```



```
CHAR buff[64];
int status;

/* Set the write timeout to 10 ticks. */
tp.tv_sec = 0;
tp.tv_nsec = 1000000000;

/* Path depends on IPC 1.1x or IPC 2.x, please refer to open() API section
   for details */
fd = open(path,O_NONBLOCK | O_WRONLY);

/* Allocate the buffer. */
status = ioctl(fd ,I_BUFFALLOC,64,&buff,"smif",&tp);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_BUFFDEALLOC

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) `ioctl` request arguments. This argument deallocates the previously allocated receive buffer.

Usage

```
int ioctl (int fildes,  
           int request,  
           void *buff)
```

Arguments

- `fildes`
Descriptor on which to perform I/O control operation.
- `request`
Selects the control operation to be performed.
- `buff`
Pointer to the buffer to be deallocated.

errno Values

- `EINVAL`
If `buff` argument is `NULL`.

Example

```
#include "services/stropts.h"  
#include "services/pipc.h"  
  
int  fd;  
int  status;  
CHAR buff[64];  
  
/* Deallocate the buffer. */  
status = ioctl(fd, I_BUFFDEALLOC, buff);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_RTIMEOUT

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) `ioctl` request arguments. This argument sets the timeout for the read from channel operation.

Usage

```
int ioctl (int                fildes,
           int                request,
           const struct timespec *timeout)
```

Arguments

- `fildes`
Descriptor with which to perform I/O control operation.
- `request`
Selects the control operation to be performed.
- `timeout`
Pointer to the structure of type `timespec` to determine the timeout for the read operation. A value of zero means no timeout. This means that read will not suspend, but it will return immediately if it is not available.

errno Values

- `EINVAL`
The timeout argument is `NULL` or specified a nanosecond value less than zero or greater than or equal to 1 billion.

Example

```
#include "services/stropts.h"
#include "services/pipc.h"

int  fd;
int  status;
CHAR buff[64];
struct timespec tp;

/* Set the write timeout to 10 ticks. */
tp.tv_sec  = 0;
tp.tv_nsec = 1000000000;

/* Deallocate the buffer. */
status = ioctl(fd, I_RTIMEOUT, &tp);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_SFLAGS

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) `ioctl` request arguments. This argument sets the flags for the corresponding channel.

Usage

```
int ioctl (int          fildes,  
           int          request,  
           unsigned char flags)
```

Arguments

- `fildes`
Descriptor on which to perform I/O control operation.
- `request`
Selects the control operation to be performed.
- `flags`
User defined flags.

Example

```
#include "services/stropts.h"  
#include "services/pipc.h"  
  
#define FLAGS 64  
  
int fd;  
int status;  
  
/* Deallocate the buffer. */  
status = ioctl(fd, I_SFLAGS, FLAGS);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_SIFFPTR

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) ioctl request arguments. This argument creates a mapping table between the name of the interface and the pointer to that interface.

Usage

```
int ioctl (int      fildes,
           int      request,
           const char *iffname,
           void      *iffptr)
```

Arguments

- **fildes**
Descriptor on which to perform I/O control operation.
- **request**
Selects the control operation to be performed.
- **iffname**
Pointer to the name of physical interface.
- **iffptr**
Pointer to the physical interface.

errno Values

- **EINVAL**
If iffname or iffptr is NULL.
- **EAGAIN**
If no resources are available in the layer to hold this interface pointer.

Example

```
#include "services/stropts.h"
#include "services/pipc.h"

/* External variable declarations. */
extern IPC_PHYSIF SM_Physif;

int fd;
int status;

/* The descriptor argument is just passed as dummy. */
status = ioctl(0, I_SIFFPTR, "smif", (void*)&SM_Physif);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

I_WTIMEOUT

Allowed from: POSIX Thread and signal handler

This is one of the Inter-Processor Communication (IPC) ioctl request arguments. This argument sets timeout for the write to channel operation.

Usage

```
int ioctl (int                fildes,  
           int                request,  
           const struct timespec *timeout)
```

Arguments

- **fildes**
Descriptor on which to perform I/O control operations.
- **request**
Selects the control operation to be performed.
- **timeout**
Pointer to the structure of type `timespec` to determine the timeout for the write operation. A value of zero means no timeout. This means that write will not suspend, but it will return immediately if it is not available.

errno Values

- **EINVAL**
The timeout argument is `NULL` or specified a nanosecond value less than zero or greater than or equal to 1 billion.

Example

```
#include "services/stropts.h"  
#include "services/pipc.h"  
  
int fd;  
int status;  
  
/* Set the write timeout to 10 ticks. */  
rqtp.tv_sec = 0;  
rqtp.tv_nsec = 1000000000;  
  
/* Set the time out for the write operation */  
status = ioctl(fd, I_WTIMEOUT, &rqtp);
```

Related Topics

[ioctl](#)

[Nucleus POSIX File System Library APIs](#)

link

Allowed from: POSIX Thread and signal handler

This function creates a new link (directory entry) for the existing file, path1. This API is currently not supported.

Usage

```
int link (const char *path1,  
          const char *path2)
```

Arguments

- path1
Pointer to the pathname naming an existing file.
- path2
Points to a pathname naming the new directory entry to be created.

Return Value

- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code. Since Nucleus file system is FAT based, the link() function will always return an error.

ENOTSUP

This function is not supported in our Nucleus POSIX implementation.

Example

```
int ret;  
  
/* currently not supported */  
ret = link("file1", "/home/cnd/mod2");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [unlink](#)


<unistd.h>

[Nucleus POSIX File System Library APIs](#)

lseek

Allowed from: POSIX Thread and signal handler

This service sets the file offset of the opened file descriptor associated with `filedes`.

 **Note** This functionality is not supported for the I/O devices.

Usage

```
off_t lseek (int    filedes,  
             off_t  offset,  
             int    whence)
```

Arguments

- `filedes`

File descriptor.

- `offset`

File offset to seek.

- `whence`

Determines from where to seek. This can be set to one of the following values:

`SEEK_SET`

The file offset is set to offset bytes.

`SEEK_CUR`

The file offset is set to offset bytes.

`SEEK_END`

The file offset is set to the size of file plus the offset. Seeking is not allowed beyond the end of the file.

Return Value

- `POSIX_SUCCESS`

Indicates successful completion of the service.

- `POSIX_ERROR`

Indicates that the service fails and sets `errno` to the corresponding error code:

`EBADF`

Bad file descriptor.

`EACCES`

Permission is denied.

EIO

I/O error.

EINVAL

Invalid argument.

Example

```
#include "services/unistd.h"

int ret, fd;
char buffer[100];

fd = open("somefile", O_RDONLY);

/* If no error */
ret = write(fd, (void*)buffer, 100);

status = lseek(fd, 0, SEEK_SET);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [open](#)

<sys/types.h>

<unistd.h>

[Nucleus POSIX File System Library APIs](#)

mkdir

Allowed from: POSIX Thread and signal handler

This service creates a new directory.

Note



In our implementation mode is not used. File permissions were not available on the directory.

Usage

```
int mkdir (const char *path,  
           mode_t      mode)
```

Arguments

- **path**
Pointer to the path where the directory is to be created.
- **mode**
Given mode of the directory to be created.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:
 - EEXIST**
File already exists.
 - EACCES**
Permission is denied.
 - EIO**
I/O error.
 - ENAMETOOLONG**
Directory name too long.
 - ENOSPC**
No space left on the device
 - ENOENT**
No such directory.

Example

```
#include "services/unistd.h"
#include "services/sys/stat.h"

int ret;

ret = mkdir("new_dir", S_IRWX);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[<sys/stat.h>](#)

[<sys/types.h>](#)

[Nucleus POSIX File System Library APIs](#)

mmap

This function establishes a mapping between a process' address space and a file.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services\sys\mman.h"

...
void *mmap(void    *addr,
            size_t  len,
            int      prot,
            int      flags,
            int      fildes,
            off_t    off)
```

Arguments

- **addr**
Specific address arguments are not supported. This should be set to zero.
- **len**
Length of the mapped memory region.
- **prot**
Protection must be (PROT_READ | PROT_WRITE) process memory is used. No changes to access permissions are supported.
- **flags**
Only MAP_SHARED is supported.
- **fildes**
File descriptor with which to create mapping.
- **off**
Offset into file where the mapping will begin for len bytes.

Return Values

- **void ***
Upon successful completion, this returns the address at which the mapping was placed.
- **MAP_FAILED**
If the call fails, it returns a value of MAP_FAILED and sets errno to the corresponding error code:

EBADF

The fildes argument is not a valid open file descriptor.

EINVAL

The value of len is zero.

EINVAL

The addr argument is considered invalid by the implementation.

EINVAL

The value of flags is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).

ENODEV

The fildes argument refers to a file whose type is not supported by mmap().

ENOTSUP

MAP_FIXED or MAP_PRIVATE was specified in the flags argument and the implementation does not support this functionality. The implementation does not support the combination of accesses requested in the prot argument.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

msync

This function writes all modified data to permanent storage locations, containing any part of the address space of the process starting at address `addr` and continuing for `len` bytes. If no such storage exists, `msync()` does not need to have any effect.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services/sys/mman.h"

...
int msync(void *addr,
          size_t len,
          int flags)
```

Arguments

- `addr`
Start address where sync will start and continuing for `len` bytes.
- `len`
All mapped regions starting at `addr` and continuing for `len` will be written to permanent storage.
- `flags`
The flags argument is constructed from the bitwise-inclusive OR of one or more of the following flags defined in the *mman.h* header:

Table 4-1. mman.h Header Files

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data. (NOT Supported)

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of this function.
- `POSIX_ERROR`
If the call fails, it returns -1 and sets `errno` to the corresponding error:
 - `EINVAL`
The value of flags is invalid.

EINVAL

The value of `addr` is not a multiple of the page size `{PAGESIZE}`.

ENOMEM

The addresses in the range starting at `addr` and continuing for `len` bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

munmap

This function removes any mappings for those entire pages containing any part of the address space of the process starting at `addr` and continuing for `len` bytes. If there are no mappings in the specified address range, then `munmap()` has no effect.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services/sys/mman.h"

...
int munmap(void *addr,
            size_t len)
```

Arguments

- `addr`
Start address where unmapping starts and continuing for `len` bytes.
- `len`
All mapped regions starting at `addr` and continuing for `len` will be unmapped.

Return Values

- `POSIX_SUCCESS`
Indicates successful completion of this function.
- `POSIX_ERROR`
If the call fails, it returns -1 and sets `errno` to the corresponding error:

`EINVAL`

Addresses in the range `[addr,addr+len)` are outside the valid range for the address space of a process.

`EINVAL`

The `len` argument is zero.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

open

Allowed from: POSIX Thread and signal handler

This service opens an existing file or creates it if does not already exist.

Usage

```
int open (const char *path,  
          int         oflags,  
          /* mode */) 
```

Arguments

- path

Pointer to the path of the file to be opened. There are three different formats for path, File I/O, Device I/O, and Nucleus IPC:

File I/O

For a description of file I/O, please refer to the previous section.

Device I/O

The case of devices path format should be */dev/devname*, where:

dev

This part should always be the same and should be used to indicate an open device.

devname

Name of the device to be opened.

Nucleus IPC

- oflags

Sets the flags on the file to be opened. Refer to [Table 4-2](#) for more information.

- mode (optional)

Optional mode of the file to be opened if the file is going to be created.

oflag Values

Table 4-2. oflag Values

Oflag	Description	Nucleus Support	IPC Support
O_RDONLY	Open for reading only.	Supported	Supported
O_WRONLY	Open for writing only.	Supported	Supported
O_RDWR	Open for reading and writing.	Supported	Supported

Table 4-2. oflag Values (cont.)

Oflag	Description	Nucleus Support	IPC Support
O_APPEND	Offset should be set to the end of the file.	Supported	Not Supported
O_CREAT	If file exists, this has no effect otherwise it creates a new file.	Supported	Not Supported
O_EXCL	If O_CREAT and O_EXCL are set and file already exists, then the call fails.	Supported	Not Supported
O_TRUNC	File is truncated to 0.	Supported	Not Supported
O_DSYNC	Write I/O operations on the file descriptor is complete as defined by synchronized I/O data integrity completion.	Not Supported	Not Supported
O_NOCTTY	If set and path identifies a terminal device, open does not cause the terminal device to become the controlling terminal for the process.	Not Supported	Not Supported
O_NONBLOCK	Not to block while performing I/O. In case of POSIX IPC if this flag is specified then this means, in the read and write APIs, we will timeout for the value passed in the structure of type timespec in the ioctl call. If it is not specified then these operations will suspend until read() or write() becomes available.	Not Supported	Supported
O_RSYNC	Read I/O operations on the file descriptor will complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags.	Supported	Not Supported
O_SYNC	Write I/O operations on the file descriptor is complete as defined by synchronized I/O file integrity completion.	Supported	Not Supported

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.

- **POSIX_ERROR**

Indicates that the service fails and sets `errno` to the corresponding error code:

ENAMETOOLONG

File name is too long.

ENOENT

No such file or directory.

EACCES

Permission is denied.

EMFILE

Maximum number of file descriptors is already open.

ENOSPC

No memory left in the system to allocate to file.

EINVAL

Invalid argument.

EINVAL

If the ASCII encoded port number passed is greater than 65535 or less than 64 in case of POSIX IPC.

EINVAL

If no interface control block is found against the interface name in case of POSIX IPC.

ENOMEM

No more memory is available to allocate memory for semaphore control block while opening a channel in case of POSIX IPC.

EEXIST

The file/channel already exists.

EIO

I/O error.

Description

This call associates a file descriptor to a file. The file offset used to mark the current position within the file is set to the beginning of the file. The file status flag and the file access modes of the open file descriptor are set according to the value of `oflag`.

Note



If the file is opened in the append mode (`O_APPEND`), then a seek is done at the beginning of the file or before the end of file, writing it will not cause it to append after the end of file. Instead it will write the current position of the file descriptor.

Example

```
#include "services/fcntl.h"

int fd;

fd = open("file1", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)

[creat](#) [dup](#)

[fcntl](#) [<fcntl.h>](#)

[ioctl](#) [lseek](#)

[read](#) [<sys/stat.h>](#)

[<sys/types.h>](#) [write](#)

[Nucleus POSIX File System Library APIs](#)

opendir

Allowed from: POSIX Thread and signal handler

This service opens a directory stream corresponding to the directory named by the `dirname` argument. The directory stream is positioned at the first entry.

Usage

```
DIR *opendir (const char *dirname)
```

Arguments

- `dirname`
Pointer to the directory name to be opened.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`ENOENT`

No such directory.

`EACCES`

Permission denied.

`ENOMEM`

Not enough space.

`ENAMETOOLONG`

Filename too long.

`EIO`

I/O error.

Example

```
#include "services/sys/types.h"
#include "services/dirent.h"

DIR *dir;

struct dirent *dp;

if ((dir = opendir (".")) == NULL)
{
    perror ("Cannot open .");
}
```

```
while ((dp = readdir (dir)) != NULL)
{
    ...
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[close](#)

[<dirent.h>](#)

[<limits.h>](#)

[readdir](#)

[rewinddir](#)

[<sys/types.h>](#)

[Nucleus POSIX File System Library
APIs](#)

read

Allowed from: POSIX Thread and signal handler

This service reads a specified number of bytes from file descriptor, `filedes`, and fills the buffer `buf`.

Usage

```
ssize_t read (int    filedes,  
              void   *buf,  
              size_t nbyte)
```

Arguments

- `filedes`
File descriptor.
- `buf`
Pointer to the buffer in which data is to be read.
- `nbyte`
Number of bytes to be read.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EBADF`

Bad file descriptor.

`EACCES`

Permission is denied.

`EIO`

I/O error.

`EINVAL`

Invalid argument.

`ETIMEDOUT`

`O_NONBLOCK` was not set when the channel was opened, but timeout occurs while reading from an IPC channel in case of POSIX IPC.

`ENOMEM`

If no memory is available in case of allocating the buffer to read from a POSIX IPC channel in case of POSIX IPC.

Description

This reads `nbyte` from the file descriptor and returns the number of successfully read bytes. If the file pointer passed the end-of-file pointer, then no read occurs. Currently, there is no support for pipes and fifo so read/write cannot be done in this case. It is also possible to read from a socket.

In our implementation this call cannot be interrupted by the signal.

Example

```
#include "services/unistd.h"

int ret, fd;
char buffer[100];

fd = open("somefile", O_RDONLY);

/* If no error */
ret = read(fd, (void*)buffer, 100);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[<dirent.h>](#)

[fcntl](#)

[ioctl](#)

[<limits.h>](#)

[lseek](#)

[open](#)

[<sys/types.h>](#)

[Nucleus POSIX File System Library APIs](#)

readdir

Allowed from: POSIX Thread and signal handler

This service takes a pointer to a directory stream and returns a pointer to the directory entry at the current position then positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. This API is non-reentrant. Refer to [readdir_r](#) for the reentrant version of this function.

Usage

```
struct dirent *readdir (DIR *dirp)
```

Arguments

- **dirp**
Pointer to the directory stream.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:

EBADF

The `dirp` argument does not refer to an open directory stream.

EACCES

Permission denied.

EIO

I/O error.

Example

```
#include "services/sys/types.h"
#include "services/dirent.h"

#define FOUND      1
#define NOT_FOUND  0
#define READ_ERROR -2
#define OPEN_ERROR -3

dirp = opendir(".");

while (dirp)
{
    errno = 0;
    if ((dp = readdir(dirp)) != NULL)
    {
        if (strcmp(dp->d_name, name) == 0)
        {
```

```
        closedir(dirp);
        return FOUND;
    }
}
else
{
    if (errno == 0)
    {
        closedir(dirp);
        return NOT_FOUND;
    }
    closedir(dirp);
    return READ_ERROR;
}

return OPEN_ERROR;
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [closedir](#)

<dirent.h> [opendir](#)

[rewinddir](#) <sys/types.h>

[Nucleus POSIX File System Library APIs](#)

readdir_r

This service takes a pointer to a directory stream and returns a pointer to the directory entry at the current position then positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. This API is reentrant. Refer to [readdir](#) for the non-reentrant version of this function.

Usage

```
#include "services\dirent.h"

...
int readdir_r(DIR          *dirp,
              struct dirent *entry,
              struct dirent **result)
```

Arguments

- **dirp**
Pointer to the directory stream.
- **entry**
Pointer to an object of type struct dirent.
- **result**
Double pointer to an object of type struct dirent.

Return Values

- **POSIX_SUCCESS**
Indicates successful completion of this function.
- **ERROR**
If the call fails, it returns the corresponding error number:

E_OVERFLOW

One of the values in the structure to be returned cannot be represented correctly.

EBADF

The dirp argument does not refer to an open directory stream.

ENOENT

The current position of the directory stream is invalid.

EBADF

The dirp argument does not refer to an open directory stream.

Description

The storage pointed to by entry is large enough for a dirent with an array of char d_name members containing at least {NAME_MAX}+1 elements.

Upon successful return, the pointer returned at *result has the same value as the argument entry. Upon reaching the end of the directory stream, this pointer has the value NULL. The readdir_r() function does not return directory entries containing empty names.

The Nucleus POSIX product and file system must be successfully initialized before calling this function.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

rename

Allowed from: POSIX Thread and signal handler

This function changes the name of a file.

Usage

```
int rename (const char *old,  
            const char *new_path)
```

Arguments

- **old**
Pointer to the pathname of file to be renamed.
- **new_path**
Pointer to the new pathname of the file.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:

EINVAL

Invalid argument.

ENAMETOOLONG

Filename too long.

ENOSPC

No space left on the device.

ENOENT

No such file.

EEXIST

File already exists.

EIO

I/O error.

EACCES

Permission denied

Description

If the `old` argument points to the pathname of a file that is not a directory, the `new_path` argument does not point to the pathname of a directory. If the link named by `new_path` exists, it

is removed and old is renamed to new_path. In this case, a link named new_path remains visible to other processes throughout the renaming operation. This link refers to either the file referred to by new_path or old before the operation began.

If old points to the pathname of a directory, new_path does not point to a file's pathname that is not a directory. If the directory named by new_path exists, it is removed and old is renamed to new_path. In this case, a link named new_path exists throughout the renaming operation and refers to either the directory referred to by new_path or old before the operation began. If new_path names an existing directory, it must be an empty directory.

Example

```
#include "services/stdio.h"

int status;

status = rename("/home/cnd/mod1.txt", "/home/cnd/mod2.tmt");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [link](#)

[rmdir](#)

[unlink](#)

<stdio.h>

[Nucleus POSIX File System Library APIs](#)

rewinddir

Allowed from: POSIX Thread and signal handler

This service resets the position of the directory stream `dirp` refers to at the beginning of the directory.

Usage

```
void rewinddir (DIR *dirp)
```

Arguments

- `dirp`
Pointer to the directory stream.

Example

```
#include "services/dirent.h"

DIR *dirp = opendir("/");

readdir(dirp);

rewinddir(dirp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	closedir
<dirent.h>	opendir
readdir	<sys/types.h>
Nucleus POSIX File System Library APIs	

rmdir

Allowed from: POSIX Thread and signal handler

This service removes the specified directory. The directory is removed only if it is an empty directory.

Usage

```
int rmdir (const char *path)
```

Arguments

- path
Pointer to the path of the directory to be removed.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and set errno to the corresponding error code:

EBUSY

Resource busy.

EACCES

Permission is denied.

EIO

I/O error.

ENAMETOOLONG

Directory name too long.

ENOTEMPTY

Directory not empty.

ENOENT

No such directory.

ENOSPC

No space left on the device.

Example

```
#include "services/unistd.h"  
#include "services/sys/stat.h"  
  
int ret;  
  
ret = mkdir("new_dir", S_IRWX);
```



```
ret = rmdir("new_dir");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[mkdir](#)

[remove](#)

[unlink](#)

[<unistd.h>](#)

[Nucleus POSIX File System Library APIs](#)

seekdir

Allowed from: POSIX Thread and signal handler

This service sets the position of the next [readdir](#) operation on the directory, at the beginning of the directory stream specified by dirp, which is the position specified by loc.

Usage

```
void seekdir (DIR *dirp,  
             long loc)
```

Arguments

- dirp
Pointer to the directory stream.
- loc
Position to seek.

Example

```
#include "services/dirent.h"  
  
DIR *dirp = opendir("/");  
  
seekdir(dirp, 0);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 <dirent.h>

[opendir](#) [readdir](#)

[telldir](#) <stdio.h>

<sys/types.h> [Nucleus POSIX File System Library APIs](#)

shm_open

This function establishes a connection between a shared memory object and a file descriptor.

Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services/sys/mman.h"
#include "services/sys/stat.h"
#include "services/fcntl.h"

...
int shm_open(const char *name,
             int         oflag,
             mode_t      mode)
```

Arguments

- name

The name argument points to a string that names a shared memory object.

- oflag

The oflag argument is the bitwise-inclusive OR of the following flags defined in the *fcntl.h* header. Applications specify exactly one of the following first two values (access modes) in the value of oflag:

O_RDONLY

Open for read access only.

O_RDWR

for read or write access.

Any combination of the remaining flags may be specified in the value of oflag:

O_CREAT

The shared memory object is created.

O_EXCL

If **O_EXCL** and **O_CREAT** are set, *shm_open()* fails if the shared memory object exists. The check for the existence of the shared memory object and the creation of the object, if it does not exist, is atomic with respect to other processes executing *shm_open()* that name the same shared memory object with the **O_EXCL** and **O_CREAT** set. If **O_EXCL** is set and **O_CREAT** is not set, the result is undefined.

O_TRUNC

the shared memory object exists, and it successfully opened **O_RDWR**, the object is truncated to zero length and the mode and owner are unchanged by this function call. The result of using **O_TRUNC** with **O_RDONLY** is undefined.

- **mode**

Sets the object's permission bits. This is defined in *services/sys/stat.h*.

Return Values

- **non-negative integer**

Upon successful completion of this function, it returns a non-negative integer representing the lowest numbered unused file descriptor

- **POSIX_ERROR**

If the call fails, it returns -1 and sets `errno` to the corresponding error:

EACCES

Permissions specified by `oflag` are denied.

EEXIST

`O_CREAT` and `O_EXCL` are set and the named shared memory object already exists.

EINTR

The `shm_open()` operation was interrupted by a signal.

EINVAL

The `shm_open()` operation is not supported for the given name.

EMFILE

Too many file descriptors are currently in use by this process.

ENAMETOOLONG

The length of the name argument exceeds `{PATH_MAX}`, or a pathname component is longer than `{NAME_MAX}`.

ENFILE

Too many shared memory objects are currently open in the system.

ENOENT

`O_CREAT` is not set and the named shared memory object does not exist.

ENOSPC

There is insufficient space for the creation of the new shared memory object.

Description

This function creates an open file description that refers to the shared memory object and a file descriptor that refers to that open file description. The file descriptor is used by other functions to refer to that shared memory object.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

shm_unlink

This function removes the name of the shared memory object named by the string pointed to by name.

Nucleus POSIX product and file system must be successfully initialized before calling this function.

Usage

```
#include "services/sys/mman.h"

...
int shm_unlink(const char *name)
```

Arguments

- name
Pointer to the name of the shared memory object.

Return Values

- 0
Indicates successful completion of this function.
- POSIX_ERROR
If the call fails, it returns -1 and sets errno to the corresponding error. In this case, the named shared memory object is not changed by this function call.
 - EACCES
Permission is denied to unlink the named shared memory object.
 - ENAMETOOLONG
The length of the name argument exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}.
 - ENOENT
The named shared memory object does not exist.

Example

Refer to IEEE 1003.1 - The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004.

Related Topics

[Nucleus POSIX File System Library APIs](#)

stat

Allowed from: POSIX Thread and signal handler

This service obtains information about the named file and writes it to the area pointed to by the buf argument.

Usage

```
int stat (const char *path,  
          struct stat *buf)
```

Arguments

- path
Pointer to the pathname naming a file.
- buf
Pointer to the buffer to be filled with the file information.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EIO
An error occurred while reading from the file system.
 - ENAMETOOLONG
The length of the path argument exceeds PATH_MAX or pathname component is longer than NAME_MAX.
 - ENOENT
A component of path does not name an existing file or path is an empty string.

Example

```
#include "services/sys/types.h"  
#include "services/sys/stat.h"  
#include "services/fcntl.h"  
  
struct stat buffer;  
int status;  
  
/* Find the information of the file "/usr/file1" */  
status = stat("/usr/file1",&buffer);
```

Related Topics

[fstat](#)

[Nucleus POSIX File System Library APIs](#)

telldir

Allowed from: POSIX Thread and signal handler

This service obtains the current location associated with the directory stream specified by dirp.

Usage

```
long telldir (DIR *dirp)
```

Arguments

- **dirp**
Pointer to the directory stream.

Example

```
#include "services/dirent.h"

long dir_offset;
DIR *dirp = opendir("/");

dir_offset = telldir(dirp);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [<dirent.h>](#)

[opendir](#)

[readdir](#)

[seekdir](#)

[Nucleus POSIX File System Library APIs](#)

unlink

Allowed from: POSIX Thread and signal handler

This service removes a link to a file.

Usage

```
int unlink (const char *path)
```

Arguments

- **path**
Pointer to the pathname of file to be unlinked.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:

EBUSY

Resource is busy.

ENAMETOOLONG

Filename too long.

ENOSPC

No space left on the device.

ENOENT

No such file.

EPERM

Operation not permitted.

EIO

I/O error.

EACCES

Permission denied.

Example

```
#include "services/unistd.h"

int ret;

ret = unlink("file1");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [close](#)
[link](#) [remove](#)
[rmdir](#) [<unistd.h>](#)
[Nucleus POSIX File System Library APIs](#)

utime

Allowed from: POSIX Thread and signal handler

This function set the access and modifications time of the file pointed to by the path argument.

Usage

```
int utime (const char          *path,  
           const struct utimbuf *times)
```

Arguments

- path
Pointer to the pathname of file to be unlinked.
- times
Pointer to the utimbuf structure.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service fails and sets errno to the corresponding error code:
 - EINVAL
If pointer to path is NULL.
 - ENAMETOOLONG
The length of the path argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
 - ENOENT
No entry is found against the given path.
 - EACCES
Permission denied.

Example

```
int ret;  
  
/* Set the access and modification time of "/usr/file1" to the current  
   time. */  
ret = utime("/usr/file1",NULL);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [fstat](#)

[stat](#)

[<sys/stat.h>](#)

[<utime.h>](#)

[Nucleus POSIX File System Library APIs](#)

write

Allowed from: POSIX Thread and signal handler

This service writes nbyte from buf to a file pointed to by a file descriptor. The write service cannot be interrupted by the signal. You can also use this to write to a socket.

Usage

```
ssize_t write (int    filedес,  
              void    *buf,  
              size_t  nbyte)
```

Arguments

- **filedes**
File descriptor.
- **buf**
Pointer to the write buffer.
- **nbyte**
Number of bytes to be written.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and set errno to the corresponding error code:

EBADF

Bad file descriptor.

EACCES

Permission is denied.

EIO

I/O error.

EINVAL

Invalid argument.

ENOSPC

No space left on the device.

EAGAIN

O_NONBLOCK flag is set while opening the channel, but writing to a channel fails because the sending interrupt is already in progress in case of POSIX IPC.

ETIMEDOUT

O_NONBLOCK was not set when the channel was opened, but the timeout occurs while writing to a channel in case of POSIX IPC.

Example

```
#include "services/unistd.h"

int ret, fd;
char buffer[100];

fd = open("somefile", O_RDONLY);

/* If no error */
ret = write(fd, (void*)buffer, 100);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[dup](#)

[ioctl](#)

[lseek](#)

[<stropts.h>](#)

[Nucleus POSIX File System Library APIs](#)

[creat](#)

[fcntl](#)

[<limits.h>](#)

[open](#)

[<unistd.h>](#)

Chapter 5

Nucleus POSIX Networking Library APIs

This chapter provides a detailed reference of all the Nucleus POSIX Networking Library (POSIX NET) APIs.

- [accept](#)
- [bind](#)
- [connect](#)
- [endhostent](#)
- [FD_CLR](#)
- [FD_ISSET](#)
- [FD_PSET](#)
- [FD_ZERO](#)
- [gethostbyaddr](#)
- [gethostbyname](#)
- [gethostent](#)
- [gethostname](#)
- [getpeername](#)
- [getsockname](#)
- [getsockopt](#)
- [htonl](#)
- [htons](#)
- [if_freenameindex](#)
- [if_indextoname](#)
- [if_nameindex](#)
- [if_nametoindex](#)
- [inet_addr](#)
- [inet_ntoa](#)

- [inet_ntop](#)
- [inet_pton](#)
- [listen](#)
- [ntohl](#)
- [ntohs](#)
- [recv](#)
- [recvfrom](#)
- [recvmsg](#)
- [select](#)
- [send](#)
- [sendmsg](#)
- [sendto](#)
- [sethostent](#)
- [setsockopt](#)
- [shutdown](#)
- [socket](#)

accept

Allowed from: POSIX Thread and signal handler

This function extracts the first connection on the queue of pending connections, creates a new socket with the same socket type protocol and address family as the specified socket, and allocates a new file descriptor for that socket.

Usage

```
int accept (int          socket,  
            struct sockaddr *address,  
            socklen_t    *address_len)
```

Arguments

- **socket**
Specifies a socket that was created with [socket](#), has been bound to an address with [bind](#), and has issued a successful call to [listen](#).
- **address**
Either a null pointer or a pointer to a `sockaddr` structure where the address of the connecting socket is returned.
- **address_len**
Points to a `socklen_t` structure which on input specifies the length of the supplied `sockaddr` structure, and on output specifies the length of the stored address.

Return Value

- **File descriptor**
New socket file descriptor.
- **POSIX_ERROR**
Indicates that the service fails and sets `errno` to the corresponding error code:
 - EWOULDBLOCK**
O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.
 - EBADF**
The socket argument is not a valid file descriptor.
 - ECONNABORTED**
A connection has been aborted.
 - EINVAL**
The socket is not accepting connections because of NULL address value or bad parameter value.
 - ENFILE**

The maximum number of file descriptors in the system are already open.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The socket type of the specified socket does not support accepting connections.

EINTR

The `accept()` function was interrupted by a signal that was caught before a valid connection arrived.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int sock, sock1;
struct sockaddr_in serv_addr;
struct sockaddr_in cli_addr;
socklen_t clilen;

sock = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(8888);

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* error */
}
else
{
    if (listen(sock, 10) < 0)
    {
        /*error*/
    }
    else
    {
        clilen = sizeof(cli_addr);
        sock1 = accept(sock, (struct sockaddr *)&cli_addr, &clilen);
    }
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [bind](#)

[connect](#) [listen](#)

[socket](#) [<sys/socket.h>](#)

[Nucleus POSIX Networking Library APIs](#)

bind

Allowed from: POSIX Thread and signal handler

This function assigns a local socket address to a socket identified by a descriptor `socket` that has no local socket address assigned. Sockets created with the [socket](#) function are initially unnamed; they are identified only by their address family.

Usage

```
int bind (int                socket,  
          const struct sockaddr *address,  
          socklen_t          address_len)
```

Arguments

- `socket`
Specifies the file descriptor of the socket to be bound.
- `address`
Points to a `sockaddr` structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
- `address_len`
Specifies the length of the `sockaddr` structure pointed to by the `address` argument.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:

`EADDRINUSE`

The specified address is already in use.

`EADDRNOTAVAIL`

The specified address is not available from the local machine.

`EAFNOSUPPORT`

The specified address is not a valid address for the address family of the specified socket.

`EBADF`

The `socket` argument is not a valid file descriptor.

`EINVAL`

The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The socket type of the specified socket does not support binding to an address.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int sock;
struct sockaddr_in serv_addr;

sock = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(8888);

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* Error */
}
else
{
    /* Success */
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [connect](#)

[getsockname](#) [listen](#)

[socket](#) <sys/socket.h>

[Nucleus POSIX Networking Library APIs](#)

connect

Allowed from: POSIX Thread and signal handler

This function attempts to make a connection on a socket.

Usage

```
int connect (int          socket,  
             const struct sockaddr *address,  
             socklen_t    address_len)
```

Arguments

- **socket**
Specifies the file descriptor associated with the socket.
- **address**
Points to a `sockaddr` structure containing the peer address. The length and format of the address depend on the address family of the socket.
- **address_len**
Specifies the length of the `sockaddr` structure pointed to by the address argument.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**

Indicates that the service fails and sets `errno` to the corresponding error code:

EADDRNOTAVAIL

The specified address is not available from the local machine.

EAFNOSUPPORT

The specified address is not a valid address for the address family of the specified socket.

EINPROGRESS

A connection request is already in progress for the specified socket.

EBADF

The socket argument is not a valid file descriptor.

ECONNREFUSED

The target address was not listening for connections or refused the connection request.

ENETUNREACH

No route to the network is present.

ENOTSOCK

The socket argument does not refer to a socket.

ETIMEDOUT

The attempt to connect timed out before a connection was made.

EADDRINUSE

Attempt to establish a connection that uses addresses that are already in use.

ECONNRESET

Remote host reset the connection request.

EINVAL

NULL address value or bad parameter value.

EINTR

The attempt to establish a connection was interrupted by delivery of a signal that was caught.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int sock;
struct sockaddr_in cli_addr;

sock = socket(AF_INET, SOCK_STREAM, 0);

if (connect(sock, (struct sockaddr *) &cli_addr,
(socklen_t)sizeof(cli_addr)) < 0)
{
    /* error */
}
else
{
    /* success */
}
```

Related Topics

accept	Base Definitions volume of IEEE Std 1003.1-2001
bind	close
getsockname	select
send	shutdown
socket	<sys/socket.h>

[Nucleus POSIX Networking Library APIs](#)

endhostent

Allowed from: POSIX Thread and signal handler

This function closes the connection to the database, releasing any open file descriptors.

Usage

```
void endhostent (void)
```

Example

```
#include "services/netdb.h"

/* close the connection to the database, releasing any open
   file descriptor. */
endhostent();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [gethostbyaddr](#)

<netdb.h>

[Nucleus POSIX Networking Library APIs](#)

FD_CLR

Allowed from: POSIX Thread and signal handler

This function removes the file descriptor, `fd`, from the set pointed to by `fdsetp`. If `fd` is not a member of this set, there is no effect on the set, and no error is returned.

Usage

```
void FD_CLR(int    fd,
            fd_set *fdsetp)
```

Arguments

- `fd`
The file descriptor to be removed from the list.
- `fdsetp`
Pointer to a set of file descriptors.

Example

```
#include "services/sys/socket.h"
#include "services/sys/select.h"

fd_set fdset;
int     sock;

FD_CLR(sock, &fdset);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[select](#)

<sys/select.h>

[Nucleus POSIX Networking Library APIs](#)

FD_ISSET

Allowed from: POSIX Thread and signal handler

This function checks if the file descriptor, `fd`, is a member of the set pointed to by `fdsetp`.

Usage

```
int FD_ISSET(int    fd,
              fd_set *fdsetp)
```

Arguments

- `fd`
The file descriptor to be evaluated.
- `fdsetp`
Pointer to a set of file descriptors.

Return Value

- non-zero
The bit for the file descriptor, `fd`, is in the file descriptor set pointed to by `fdset`.
- 0
The bit for the file descriptor, `fd`, is not in the file descriptor set pointed to by `fdset`.

Example

```
#include "services/sys/socket.h"
#include "services/sys/select.h"

fd_set fdset;
int sock;

FD_ISSET(sock, &fdset)
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [select](#)

<sys/select.h>

[Nucleus POSIX Networking Library APIs](#)

FD_PSET

Allowed from: POSIX Thread and signal handler

This function adds the file descriptor, `fd`, to the set pointed to by `fdsetp`. If the file descriptor `fd` is already in this set, there is no effect on the set and no error is returned.

The `FD_PSET` function is used instead of the `FD_SET` function because of a name space conflict with Nucleus NET.

Usage

```
void FD_PSET(int    fd,
              fd_set *fdsetp)
```

Arguments

- `fd`
The file descriptor to be set.
- `fdsetp`
Pointer to a set of file descriptors.

Example

```
#include "services/sys/socket.h"
#include "services/sys/select.h"

fd_set fdset;
int sock;

FD_PSET(sock, &fdset);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [select](#)

<sys/select.h> [Nucleus POSIX Networking Library APIs](#)

FD_ZERO

Allowed from: POSIX Thread and signal handler

This function initializes the descriptor set pointed to by `fdsetp` to the null set. No error is returned if the set is not empty at the time `FD_ZERO()` is invoked.

Usage

```
void FD_ZERO(fd_set *fdsetp)
```

Arguments

- `fdsetp`
Pointer to a set of file descriptors

Example

```
#include "services/sys/socket.h"
#include "services/sys/select.h"

fd_set fdset;

FD_ZERO(&fdset);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[select](#)

<sys/select.h>

[Nucleus POSIX Networking Library APIs](#)

gethostbyaddr

Allowed from: POSIX Thread and signal handler

This function retrieves information about hosts. The `gethostbyaddr` function returns an entry containing addresses of the address family type for the host with address `addr`. This function is non-reentrant.

Usage

```
struct hostent *gethostbyaddr (const void *addr,  
                               socklen_t  len,  
                               int         type)
```

Arguments

- `addr`
It contains a binary format address in network byte order. (It will be an `in_addr` structure when type is `AF_INET`).
- `len`
The `len` argument contains the length of the address pointed to by `addr`.
- `type`
Address family type.

Return Value

- Pointer
Indicates successful completion of the service that returns the pointer to `hostent` structure.
- `NULL`
Entry not found.

Example

```
#include "services/netdb.h"  
#include "services/stdio.h"  
  
struct in_addr addr;  
struct hostent *hd;  
  
addr.s_addr = inet_addr("198.100.100.0");  
hd = gethostbyaddr(&(addr.s_addr), sizeof(addr), AF_INET);  
  
printf("Host name: %s\n", hd->h_name);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [endhostent](#)

[getaddrinfo](#) [inet_addr](#)

<netdb.h>

Nucleus POSIX Networking Library APIs

gethostbyname

Allowed from: POSIX Thread and signal handler

This function retrieves information about hosts. The `gethostbyname` function returns an entry containing addresses of address family `AF_INET` for the host with name `name`. This function is non-reentrant.

Usage

```
struct hostent *gethostbyname (const char *name)
```

Arguments

- `name`
The host name.

Return Value

- Pointer
Indicates successful completion of the service that returns the pointer to `hostent` structure.
- `NULL`
Entry not found.

Example

```
#include "services/netdb.h"
#include "services/stdio.h"

struct hostent *he;

he = gethostbyname("GLEN");
if (he == NULL)
    perror("gethostbyname");
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [endhostent](#)

[getaddrinfo](#) [inet_addr](#)

[<netdb.h>](#) [Nucleus POSIX Networking Library APIs](#)

gethostent

Allowed from: POSIX Thread and signal handler

This function reads the next entry in the database, opening and closing a connection to the database as necessary.

Usage

```
struct hostent *gethostent (void)
```

Return Value

- **Pointer**
Indicates successful completion of the service that returns the pointer to hostent structure.
- **NU_NULL**
Entry not found.

Example

```
#include "services/netdb.h"

struct hostent *he;

he = gethostent();
while(he != NU_NULL)
    he = gethostent();
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[gethostbyaddr](#)

<netdb.h>

[Nucleus POSIX Networking Library APIs](#)

gethostname

Allowed from: POSIX Thread and signal handler

This function returns the standard host name for the current machine.

Usage

```
int gethostname (char    *name,  
                size_t  namelen)
```

Arguments

- **name**
The name of the current host.
- **namelen**
Specify the size of the array pointed to by the name argument.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service has failed.

Example

```
#include "services/unistd.h"  
#include "services/stdio.h"  
  
char hostname[HOST_NAME_MAX];  
int  sts;  
  
/* Get the hostname */  
sts = gethostname(hostname, sizeof(hostname));  
if (sts != POSIX_SUCCESS)  
    printf("gethostname failed \n");  
else  
    printf("host name: %s\n", hostname);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1- [utime](#)
2001

<unistd.h>

[Nucleus POSIX Networking Library APIs](#)

getpeername

Allowed from: POSIX Thread and signal handler

This function retrieves the peer address of the specified socket, stores this address in the `sockaddr` structure pointed to by the `address` argument, and stores the length of this address in the object pointed to by the `address_len` argument.

Usage

```
int getpeername (int          socket,  
                 struct sockaddr *restrict address,  
                 socklen_t *restrict address_len)
```

Arguments

- `socket`
Specifies the file descriptor of the socket.
- `address`
Points to a `sockaddr` structure containing the peer address of the specified socket address.
- `address_len`
Specifies the length of the `sockaddr` structure pointed to by the `address` argument.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service has failed and sets `errno` to the corresponding error code:

`EBADF`

The `socket` argument is not a valid file descriptor.

`EINVAL`

The socket has been shut down or `NULL` address value or bad parameter value.

`ENOTCONN`

The socket is not connected or otherwise has not had the peer pre-specified.

`ENOTSOCK`

The `socket` argument does not refer to a socket.

`EOPNOTSUPP`

The operation is not supported for the socket protocol.

`ENOBUFS`

Insufficient resources were available in the system to complete the call.

Example

```
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

socklen_t size;
struct sockaddr_in cli_addr;

size = (socklen_t)sizeof(struct sockaddr);

/* Get the peer address name */
getpeername(socket, (struct sockaddr *)&cli_addr, &size);
```

Related Topics

[accept](#)

Base Definitions volume of IEEE Std 1003.1-2001

[bind](#)

[getsockname](#)

[socket](#)

<sys/socket.h>

[Nucleus POSIX Networking Library APIs](#)

getsockname

Allowed from: POSIX Thread and signal handler

This function retrieves the locally-bound name of the specified socket, stores this address in the `sockaddr` structure pointed to by the `address` argument, and stores the length of this address in the object pointed to by the `address_len` argument.

Usage

```
int getsockname (int          socket,  
                 struct sockaddr *address,  
                 socklen_t    *address_len)
```

Arguments

- `socket`
Specifies the file descriptor associated with the socket.
- `address`
Points to a `sockaddr` structure containing
- `address_len`
Specifies the length of the `sockaddr` structure pointed to by the `address` argument.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code;
 - `EBADF`
The `socket` argument is not a valid file descriptor.
 - `ENOTSOCK`
The `socket` argument does not refer to a socket.
 - `EOPNOTSUPP`
The operation is not supported for this socket's protocol.
 - `EINVAL`
The socket has been shut down, or it has a `NULL` address value or bad parameter value.

Example

```
#include "services/inet/in.h"  
#include "services/arpa/inet.h"  
#include "services/sys/socket.h"  
  
struct sockaddr_in cli_addr;
```

```
size_t addrlen;
int sock;

/* Create socket */
...

addrlen = sizeof(struct sockaddr_in);

if (getsockname(sock, (struct sockaddr *)&cli_addr, &addrlen) ==
    POSIX_ERROR)
{
    /* Error */
}
else
{
    /* Success */
}
```

Related Topics

[accept](#)

Base Definitions volume of IEEE Std 1003.1-2001

[bind](#)

[getpeername](#)

[socket](#)

<sys/socket.h>

[Nucleus POSIX Networking Library APIs](#)

getsockopt

Allowed from: POSIX Thread and signal handler

This function manipulates options associated with a socket.

Usage

```
int getsockopt (int      socket,  
               int      level,  
               int      option_name,  
               void     *option_value,  
               socklen_t *option_len)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **level**
Specifies the protocol level at which the option resides. Valid levels are SOL_SOCKET, IPPROTO_IP, IPPROTO_IPV6, and IPPROTO_TCP. Refer to [Table 5-1](#) for more information.

Table 5-1. getsockopt Levels

Level	Family	Meaning
SOL_SOCKET		
SO_BROADCAST	IPv4	Checks the broadcast status of a socket. When a socket is created the ability to send broadcasts is enabled by default. Upon a successful return the SO_BROADCAST bit will be set in the optval parameter.
SO_LINGER	IPv4/IPv6	Determines whether the linger option is enabled on the socket.
SO_REUSEADDR	IPv4/IPv6	Determines whether the option to reuse a port number for a different IP address is set on the socket.
SO_ACCEPTCONN	IPv4/IPv6	Reports whether socket listening is enabled. This option stores an int value. This is a Boolean option.
SO_TYPE	IPv4/IPv6	Reports the socket type. This option stores an int value.
SO_ERROR	IPv4/IPv6	Reports information about error status and clears it. This option stores an int value.
IPPROTO_IP		

Table 5-1. getsockopt Levels (cont.)

Level	Family	Meaning
IP_MULTICAST_TTL	IPv4	Check the TTL (Time To Live) value that is used for multicast packets that are sent using this socket. The default TTL is 1. This keeps routers from forwarding the multicast datagrams beyond the local network. Upon successful completion the optval parameter will contain the multicast TTL for the socket.
IP_MULTICAST_IF	IPv4	Check which interface has been set as the interface for sending multicast datagrams. The IP address of the interface is returned in the optval parameter as a 4 byte array.
IP_BROADCAST_IF	IPv4	Check which interface has been set as the interface for sending broadcast datagrams. The IP address of the interface is returned in the optval parameter as a 4 byte array.
IPPROTO_IPV6		
IP_RECVIFADDR	IPv4	Check if the receive interface address option has been enabled. This option enables the storing of the receive interface upon calls to NU_Recv_From . After a call to NU_Recv_From() a call to NU_Recv_IF_Addr can be made in order to obtain the IP address of the interface which received the datagram. This is useful when receiving multicast and broadcast packets that do not have a specific destination IP address.
IP_HDRINCL	IPv4	This option is only valid for IPv4 Rawsockets. When set, this option specifies that the application layer is responsible for filling in the IPv4 header. When clear the application will only pass data to the stack, which will then append an IPv4 header. This option is only valid when using the raw IP services.
IP_TTL	IPv4	Retrieve the IPv4 Time To Live for the specified socket.
IPV6_CHECKSUM	IPv6	Retrieves the value of the socket option for enabling or disabling the socket option for the IP layer to compute the checksum for outgoing IPv6 RAW packets and verify the checksum for incoming IPv6 RAW packets on the socket.

Table 5-1. getsockopt Levels (cont.)

Level	Family	Meaning
IP_TOS	IPv4	Retrieve the Type of Service for the specified IPv4 socket. ToS is the historic name for this parameter but is really used as Diff Service Code Point (DSCP).
IPV6_DSTOPTS	IPv6	Retrieves the value of the Destination Options Sticky Option that was previously set by the application.
IPV6_HOPOPTS	IPv6	Retrieves the value of the Hop-By-Hop Options Sticky Option that was previously set by the application.
IPV6_MULTICAST_HOPS	IPv6	Retrieves the value of the hop limit used for outgoing multicast packets on the socket.
IPV6_MULTICAST_IF	IPv6	Retrieves the value of the interface used for outgoing multicast packets on the socket.
IPV6_NEXTHOP	IPv6	Retrieves the value of the Next-Hop Sticky Option that was previously set by the application.
IPV6_PKTINFO	IPv6	Retrieves the value of the source address and outgoing interface index Sticky Option that was previously set by the application.
IPV6_RECVDSTOPTS	IPv6	Returns the value of the socket option to receive any Destination Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPLIMIT	IPv6	Returns the value of the socket option to receive the Hop Limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPOPTS	IPv6	Returns the value of the socket option to receive any Hop-By-Hop Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

Table 5-1. getsockopt Levels (cont.)

Level	Family	Meaning
IPV6_RECVPKTINFO	IPv6	Returns the value of the socket option to receive the receive interface and destination address of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVRTHDR	IPv6	Returns the value of the socket option obtrusive the Routing Header for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVTCLASS	IPv6	Retrieves the value of the socket option for enabling or disabling the socket option to return the Traffic Class of the most recently received IPv6 packet on a socket. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RTHDR	IPv6	Retrieves the value of the routing header Sticky Option that was previously set by the application.
IPV6_RTHDRDSTOPTS	IPv6	Retrieves the value of the Destination Options Sticky Option for Destination Options that precede the Routing Header that was previously set by the application.
IPV6_TCLASS	IPv6	Retrieves the value of the Traffic Class Sticky Option that was previously set by the application.
IPV6_V6ONLY	IPv6	Retrieves the value of the socket option to receive IPv6 packets only on a socket of family type NU_FAMILY_IP6.
IPV6_UNICAST_HOPS	IPv6	Retrieves the value of the hop limit used in outgoing unicast IPv6 packets.

Table 5-1. getsockopt Levels (cont.)

Level	Family	Meaning
TCP_NODELAY	IPv4/IPv6	TCP sockets, by default, utilize the Nagle Algorithm. The Nagle Algorithm attempts to merge contiguous small TCP packets into a single large TCP packet. It does this by delaying the transmission of small packets. This better utilizes the network bandwidth. However, it is often desirable for data to be transmitted immediately. The TCP_NODELAY option can be used to retrieve the current disposition of the Nagle Algorithm. A value of 1 indicates no delay, i.e., the Nagle Algorithm is off. A value of 0 indicates small packets will be delayed, i.e., the Nagle Algorithm is on.
SO_KEEPALIVE	IPv4/IPv6	This option determines whether TCP Keep-Alive has been enabled on the socket.
IPPROTO_TCP Level		
SO_KEEPALIVE	IPv4/IPv6	Enable TCP Keep-Alive on this socket.
TCP_NODELAY	IPv4/IPv6	TCP sockets, by default, utilize the Nagle Algorithm. The Nagle Algorithm attempts to merge contiguous small TCP packets into a single large TCP packet. It does this by delaying the transmission of small packets. This better utilizes the network bandwidth. However, it is often desirable for data to be transmitted immediately. The TCP_NODELAY option can be used to control the disposition of the Nagle Algorithm. A value of 1 will eliminate the delay, for example, turn the Nagle Algorithm off. A value of 0 will enable the delay, for example, turn the Nagle Algorithm on.
Types	Level	Support
SO_DEBUG	N/A	NOT SUPPORTED
SO_OOINLINE	N/A	NOT SUPPORTED
SO_SNDBUF	N/A	NOT SUPPORTED
SO_RCVBUF	N/A	NOT SUPPORTED
SO_DONTROUTE	N/A	NOT SUPPORTED
SO_RCVLOWAT	N/A	NOT SUPPORTED

Table 5-1. getsockopt Levels (cont.)

Level	Family	Meaning
SO_RCVTIMEO	N/A	NOT SUPPORTED
SO_SNDLOWAT	N/A	NOT SUPPORTED
SO_SNDTIMEO	N/A	NOT SUPPORTED

- option_name

Specifies a single option to be retrieved. This can be set to any of the following values. Refer to [Table 5-1](#) for more information.

SO_BROADCAST	SO_LINGER	IP_HDRINCL IP_TTL
IP_TOS	IP_MULTICAST_TTL	IP_MULTICAST_IF
IP_BROADCAST_IF	IP_RECVIFADDR	TCP_NODELAY
SO_KEEPALIVE	SO_REUSEADDR	IPV6_CHECKSUM
IPV6_DSTOPTS	IPV6_HOPOPTS	IPV6_MULTICAST_HOPS
IPV6_MULTICAST_IF	IPV6_NEXTHOP	IPV6_PKTINFO
IPV6_RECVDSTOPTS	IPV6_RECVHOPLIMIT	IPV6_RECVHOPOPTS
IPV6_RECVPKTINFO	IPV6_RECVRTHDR	IPV6_RECVTCLASS
IPV6_RTHDR	IPV6_RTHDRDSTOPTS	IPV6_TCLASS
IPV6_UNICAST_HOPS	IPV6_V6ONLY	

- option_value

The option value of the option length.

- option_len

Pointer to the length of the option_value in bytes.

Note



Refer to the [NET](#) chapter in the *Nucleus Networking Guide* for option_value data structure that is outside IEEE POSIX specification. The [setsockopt](#) API can be used to set option values based on the level specified within Nucleus NET.

Return Value

- POSIX_SUCCESS

Indicates successful completion of the service.

- POSIX_ERROR

Indicates that the service fails and sets errno to the corresponding error code:

EBADF

The socket argument is not a valid file descriptor.

EINVAL

The specified option is invalid at the specified socket level.

ENOPROTOOPT

The option is not supported by the protocol.

ENOTSOCK

The socket argument does not refer to a socket.

Description

This function retrieves the value for the option specified by the `option_name` argument for the socket specified by the `socket` argument. If the size of the option value is greater than `option_len`, the value stored in the object pointed to by the `option_value` argument is silently truncated. Otherwise, the object pointed to by the `option_len` argument is modified to indicate the actual length of the value.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int setval;

/* Check to see if the SO_BROADCAST flag is set */
getsockopt(sock, SOL_SOCKET, SO_BROADCAST, &setval, &setval);
if (setval != 0)
{
    /* SO_BROADCAST is now set to be enabled on socket sock */
}
/* setval data is different depending on the level and option name.
   Please refer to Nucleus Networking Guide for option name outside IEEE
   POSIX specification */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[bind](#)

[close](#)

[endprotoent](#)

[<netinet/in.h>](#)

[setsockopt](#)

[socket](#)

[<sys/socket.h>](#)

[Nucleus POSIX Networking Library APIs](#)

htonl

Allowed from: POSIX Thread and signal handler

This function converts the unsigned integer hostlong from host byte order to network byte order. It returns the argument value converted from host to network long byte order.

Usage

```
uint32_t htonl (uint32_t hostlong)
```

Arguments

- `hostlong`
32-bit quantity long unsigned integer.

Return Value

- Non-zero value
Conversion succeeds.

Example

```
#include "services/arpa/inet.h"

struct sockaddr_in cli_addr;

cli_addr.sin_family      = AF_INET;
cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

Related Topics

accept	<arpa/inet.h>
Base Definitions volume of IEEE Std 1003.1-2001	connect
endhostent	<inttypes.h>
sendto	Nucleus POSIX Networking Library APIs

hton

Allowed from: POSIX Thread and signal handler

This function converts the unsigned short integer hostshort from host byte order to network byte order. It returns the argument value converted from host to network short byte order.

Usage

```
uint16_t hton (uint16_t hostshort)
```

Arguments

- **hostshort**
16-bit quantity short unsigned integer.

Return Value

- **Non-zero value**
Conversion succeeds.

Example

```
#include "services/arpa/inet.h"

struct sockaddr_in cli_addr;
uint16_t portnumber = 900;

cli_addr.sin_family = AF_INET;
cli_addr.sin_port   = hton(portnumber);
```

Related Topics

accept	<arpa/inet.h>
Base Definitions volume of IEEE Std 1003.1-2001	connect
endhostent	<inttypes.h>
sendto	Nucleus POSIX Networking Library APIs

if_freenameindex

Allowed from: POSIX Thread and signal handler

This function frees the memory allocated by the [if_nameindex](#) function.

Usage

```
void if_freenameindex (struct if_nameindex *ptr)
```

Arguments

- ptr

A pointer that was returned by the [if_nameindex](#) function.

Example

```
#include "services/net/if.h"

/* Free the memory */
if_freenameindex(tmp_ptr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getsockopt
if_indextoname	if_nameindex
if_nametoindex	<net/if.h>
setsockopt	Nucleus POSIX Networking Library APIs

if_indextoname

Allowed from: POSIX Thread and signal handler

This function maps an interface index to its corresponding name.

Usage

```
char *if_indextoname (unsigned ifindex,  
                     char      *ifname)
```

Arguments

- ifindex
Interface index.
- ifname
ifname points to a buffer of at least {IF_NAMESIZE} bytes.

Return Value

- Char
pointer to buffer now containing the interface name.
- NU_NULL
Entry not found.

Example

```
#include "services/net/if.h"  
  
char if_name[IF_NAMESIZE];  
unsigned if_index = 1;  
  
if (if_indextoname(if_index, if_name) != NU_NULL)  
{  
    ...  
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getsockopt
if_freenameindex	if_nameindex
if_nametoindex	<net/if.h>
setsockopt	Nucleus POSIX Networking Library APIs

if_nameindex

Allowed from: POSIX Thread and signal handler

This function returns an array of if_nameindex structures, one structure per interface.

Usage

```
struct if_nameindex *if_nameindex (void)
```

Return Value

- **if_nameindex**
An array of structures identifying local interfaces.
- **NU_NULL**
Entry not found.

Example

```
#include "services/net/if.h"

struct if_nameindex *ni_ptr, *tmp_ptr;

/* Get a list of interfaces */

ni_ptr = if_nameindex();
tmp_ptr = ni_ptr;

/* Traverse the list */
while ( (ni_ptr->if_index) || (ni_ptr->if_name) )
{
    ni_ptr = (struct if_nameindex *)
        ((char*)ni_ptr + sizeof(struct if_nameindex) + IF_NAMESIZE);
    . . .
}

/* Free the memory */
if_freenameindex(tmp_ptr);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [getsockopt](#)

[if_freenameindex](#)

[if_indextoname](#)

[if_nametoindex](#)

[<net/if.h>](#)

[setsockopt](#)

[Nucleus POSIX Networking Library APIs](#)

if_nametoindex

Allowed from: POSIX Thread and signal handler

This function returns the interface index corresponding to name ifname.

Usage

```
unsigned if_nametoindex (const char *ifname)
```

Arguments

- ifname
name of an interface.

Return Value

- indexIf
If name is the name of an interface.
- 0
Entry not found.

Example

```
#include "services/net/if.h"

char if_name[] = "eth0_dev";
int if_index;

if_index = if_nametoindex(if_name);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[getsockopt](#)

[if_freenameindex](#)

[if_indextoname](#)

[if_nameindex](#)

[<net/if.h>](#)

[setsockopt](#)

[Nucleus POSIX Networking Library APIs](#)

inet_addr

Allowed from: POSIX Thread and signal handler

This function converts the string pointed to by `cp`, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address. This function manipulates IPv4 addresses.

Usage

```
in_addr_t inet_addr (const char *cp)
```

Arguments

- `cp`
String pointed in the standard IPv4 dotted decimal notation.

Return Value

- Non-zero value
Internet address.
- `POSIX_ERROR`
Indicates that the service fails and returns `(in_addr_t)(-1)`.

Example

```
#include "services/arpa/inet.h"
#include "services/stdio.h"

struct sockaddr_in serv_addr;
char test[] = "127.0.0.1";

serv_addr.sin_family      = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(test);
serv_addr.sin_port        = htons(8888);
```

Related Topics

[accept](#) [<arpa/inet.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001 [connect](#)

[endhostent](#) [sendto](#)

[Nucleus POSIX Networking Library APIs](#)

inet_ntoa

Allowed from: POSIX Thread and signal handler

This function converts the Internet host address specified by into a string in the Internet standard dot notation. This function manipulates IPv4 addresses.

Usage

```
char *inet_ntoa (struct in_addr in)
```

Arguments

- **in**
Internet host address.

Return Value

- **Pointer**
Pointer to the network address in Internet standard dot notation.

Example

```
#include "services/arpa/inet.h"
#include "services/stdio.h"

char *str_ptr;
struct sockaddr_in cli_addr;

/* IP address in a.b.c.d format for IPv4 */
str_ptr = inet_ntoa(cli_addr.sin_addr);
printf("Source IP:  %s\n",str_ptr);
```

Related Topics

[accept](#) [<arpa/inet.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001 [connect](#)

[endhostent](#) [sendto](#)

[Nucleus POSIX Networking Library APIs](#)

inet_ntop

Allowed from: POSIX Thread and signal handler

This function converts a numeric IPv4 or IPv6 address to a text string suitable for presentation.

Usage

```
const char *inet_ntop (int      af,  
                       const void *src,  
                       char      *dst,  
                       socklen_t size)
```

Arguments

- **af**
Specifies the family of the address.
- **src**
Points to a buffer holding an address.
- **dst**
Points to a buffer where the function stores the resulting text string; it is not be NULL.
- **size**
Specifies the size of this buffer, which is large enough to hold the text string.

Return Value

- **Pointer**
Pointer to the buffer containing the text string.
- **NULL**
Returns NULL and sets errno to the corresponding error code:

EAFNOSUPPORT

The af argument is invalid.

ENOSPC

The size of the inet_ntop() result buffer is inadequate.

Example

```
#include "services/arpa/inet.h"  
  
struct sockaddr_in cli_addr;  
char *ptr;  
char str[INET_ADDRSTRLEN];  
  
/* Ipv4 example */  
ptr = inet_ntop(AF_INET, &cli_addr.sin_addr, str, sizeof(str));
```

Related Topics

[accept](#)

[<arpa/inet.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[connect](#)

[sendto](#)

[Nucleus POSIX Networking Library APIs](#)

inet_pton

Allowed from: POSIX Thread and signal handler

This function converts a textual IPv4 or IPv6 address to a numeric binary form.

Usage

```
int inet_pton (int      af,  
               const char *src,  
               void      *dst)
```

Arguments

- **af**
Specifies the family of the address.
- **src**
Points to a buffer holding an address.
- **dst**
Points to a buffer where the function stores the resulting text string; it is not NULL.

Return Value

- 1
Conversion succeeds.
- 0
The input is not a valid IPv4 dotted-decimal string [IP6] or a valid IPv6 address string.
- **POSIX_ERROR**
If the call fails, it returns -1 and sets `errno` to the corresponding error:
 - EAFNOSUPPORT**
The `af` argument is invalid.
 - ENOSPC**
The size of the `inet_ntop()` result buffer is inadequate.

Example

```
#include "services/arpa/inet.h"  
  
char str[INET_ADDRSTRLEN];  
struct sockaddr_in cli_addr;  
  
/* Ipv4 example */  
inet_pton(AF_INET, str, &cli_addr.sin_addr);
```


Related Topics

[accept](#)

[<arpa/inet.h>](#)

Base Definitions volume of IEEE Std 1003.1-2001

[connect](#)

[sendto](#)

[Nucleus POSIX Networking Library APIs](#)

listen

Allowed from: POSIX Thread and signal handler

This function marks a connection-mode socket, specified by the `socket` argument, as accepting connections.

Usage

```
int listen (int socket,  
           int backlog)
```

Arguments

- `socket`
Specifies the file descriptor associated with the socket.
- `backlog`
Provides a hint to the implementation which the implementation uses to limit the number of outstanding connections in the socket's listen queue.

Return Value

- `POSIX_SUCCESS`
Indicates successful completion of the service.
- `POSIX_ERROR`
Indicates that the service fails and sets `errno` to the corresponding error code:
 - `EBADF`
The `socket` argument is not a valid file descriptor.
 - `EDESTADDRREQ`
The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.
 - `EINVAL`
The socket is already connected.
 - `ENOTSOCK`
The `socket` argument does not refer to a socket.
 - `EOPNOTSUPP`
The socket protocol does not support `listen()`.

Example

```
#include "services/inet/in.h"  
#include "services/arpa/inet.h"  
#include "services/sys/socket.h"  
  
int sock;  
struct sockaddr_in serv_addr;
```

```
sock          = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port   = htons(8888);

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* error */
}
else
{
    if (listen(sock, 10) < 0)
    {
        /* error */
    }
    else
    {
        /* success */
    }
}
```

Related Topics

[accept](#)

Base Definitions volume of IEEE Std 1003.1-2001

[connect](#)

[socket](#)

[<sys/socket.h>](#)

[Nucleus POSIX Networking Library APIs](#)

ntohl

Allowed from: POSIX Thread and signal handler

This function converts the unsigned integer netlong from network byte order to host byte order. It returns the argument value converted from network to host long byte order.

Usage

```
uint32_t ntohl (uint32_t netlong)
```

Arguments

- `netlong`
32-bit quantity long unsigned integer.

Return Value

- Non-zero value
Conversion succeeds.

Example

```
#include "services/arpa/inet.h"

uint32_t in;
uint32_t out;

out = ntohl(in);
```

Related Topics

accept	<arpa/inet.h>
Base Definitions volume of IEEE Std 1003.1-2001	connect
endhostent	<inttypes.h>
sendto	Nucleus POSIX Networking Library APIs

ntohs

Allowed from: POSIX Thread and signal handler

This function converts the unsigned short integer netshort from network byte order to host byte order. It returns the argument value converted from network to host short byte order.

Usage

```
uint16_t ntohs (uint16_t netshort)
```

Arguments

- **netshort**
16-bit quantity short unsigned integer.

Return Value

- **Non-zero value**
Conversion succeeds.

Example

```
#include "services/arpa/inet.h"

uint16_t in;
uint16_t out;

out = ntohs(in);
```

Related Topics

accept	<arpa/inet.h>
Base Definitions volume of IEEE Std 1003.1-2001	connect
endhostent	<inttypes.h>
sendto	Nucleus POSIX Networking Library APIs

recv

Allowed from: POSIX Thread and signal handler

This function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

Usage

```
ssize_t recv (int      socket,  
              void     *buffer,  
              size_t   length,  
              int      flags)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **buffer**
Points to a buffer where the message should be stored.
- **length**
Specifies the length in bytes of the buffer pointed to by the buffer argument.
- **flags**
Specifies the type of message reception. Refer to [Table 5-2](#) for more information.

Table 5-2. recv Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.
MSG_WAITALL	SUPPORTED	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

Return Value

- **Number of bytes**
The length of the message in bytes.

- 0

No messages are available to be received and the peer has performed an orderly shutdown.

- POSIX_ERROR

Indicates that the service fails and sets `errno` to the corresponding error code:

EWouldBlock

The socket's file descriptor is marked `O_NONBLOCK` and no data is waiting to be received; or `MSG_OOB` is set and no out-of-band data is available and either the socket's file descriptor is marked `O_NONBLOCK` or the socket does not support blocking to await out-of-band data.

EBADF

The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

ENOTCONN

A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The specified flags are not supported for this socket type or protocol.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

EMSGSIZE

The message is too large to be sent all at once, as the socket requires.

EINTR

The `recv()` function was interrupted by a signal that was caught before any data was available.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int sock, sock1;
struct sockaddr_in serv_addr;
struct sockaddr_in cli_addr;
socklen_t clilen;
ssize_t msg;
char buffer[2500];
```

```
memset(buffer, 'A', 2500);

sock          = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family   = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port     = htons(8888);

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* error */
}
else
{
    if (listen(sock, 10) < 0)
    {
        /* error */
    }
    else
    {
        clilen = sizeof(cli_addr);
        sock1 = accept(sock, (struct sockaddr *)&cli_addr, &clilen);

        /* Block until message arrives */
        msg = recv(sock1, buffer, sizeof(buffer), 0);
    }
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [read](#)

[recvfrom](#) [recvmsg](#)

[select](#) [send](#)

[sendmsg](#) [sendto](#)

[shutdown](#) [socket](#)

[<sys/socket.h>](#) [write](#)

[Nucleus POSIX Networking Library APIs](#)

recvfrom

Allowed from: POSIX Thread and signal handler

This function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

Usage

```
ssize_t recvfrom (int          socket,  
                  void         *buffer,  
                  size_t       length,  
                  int          flags,  
                  struct sockaddr *address,  
                  socklen_t     *address_len)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **buffer**
Pointer to the buffer where the message is stored.
- **length**
Specifies the length in bytes of the buffer pointed to by the buffer argument.
- **flags**
Specifies the type of message reception. Refer to [Table 5-3](#) for more information.

Table 5-3. recvfrom Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.
MSG_WAITALL	SUPPORTED	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

- **address**
A null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.

- **address_len**
Specifies the length of the sockaddr structure pointed to by the address argument.

Return Value

- **Number of bytes**
The length of the message in bytes.
- **0**
No messages are available to be received and the peer has performed an orderly shutdown.
- **POSIX_ERROR**

Indicates that the service fails and sets errno to the corresponding error code:

EWOULDBLOCK

The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF

The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EINTR

A signal interrupted recvfrom() before any data was available.

ENOTCONN

A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The specified flags are not supported for this socket type.

EINTR

A signal interrupted recvfrom() before any data was available.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

struct sockaddr_in serv_addr, cli_addr;
int sock;
socklen_t len;
ssize_t msg;
```

```
char buffer[2500];

memset(buffer, 'A', 2500);

sock          = socket(AF_INET, SOCK_DGRAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port   = htons(8888);

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* error */
}
else
{
    len = sizeof(cli_addr);
    msg = recvfrom(sock, buffer, sizeof(buffer), 0,
                   (struct sockaddr *)&cli_addr, &len);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [read](#)

[recv](#) [recvmsg](#)

[select](#) [send](#)

[sendmsg](#) [sendto](#)

[shutdown](#) [socket](#)

[<sys/socket.h>](#) [write](#)

[Nucleus POSIX Networking Library APIs](#)

recvmsg

Allowed from: POSIX Thread and signal handler

This function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

Usage

```
ssize_t recvmsg (int          socket,  
                 struct msghdr *message,  
                 int          flags)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **message**
Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The msg_flags member is ignored on input, but may contain meaningful values on output.
- **flags**
Specifies the type of message reception. Refer to [Table 5-4](#) for more information.

Table 5-4. recvmsg Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.
MSG_WAITALL	SUPPORTED	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

Return Value

- Number of bytes
The length of the message in bytes.

- 0

No messages are available to be received and the peer has performed an orderly shutdown.

- POSIX_ERROR

Indicates that the service fails and sets errno to the corresponding error code:

EWOULDBLOCK

The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF

The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EINTR

This function was interrupted by a signal before any data was available.

EINVAL

The sum of the iov_len values overflows a ssize_t.

EMSGSIZE

The message is too large to be sent all at once, as the socket requires.

ENOTCONN

A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The specified flags are not supported for this socket type.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

EINTR

This function was interrupted by a signal before any data was available.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

struct sockaddr_in serv_addr;
int sock;
```

```
ssize_t msg;
struct msghdr msgheader;
struct iovec iov[3];

/* Use Nucleus NET ancillary data type definition */
cmshdr mhead;

/* Set UDP socket */
sock = socket(AF_INET, SOCK_DGRAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(8888);

/* Specify the components of the message in an "iovec". */
iov[0].iov_base = &buf3;
iov[0].iov_len = sizeof(buf3);
iov[1].iov_base = &buf4;
iov[1].iov_len = sizeof(buf4);
iov[2].iov_base = &buf5;
iov[2].iov_len = sizeof(buf5);

msgheader.msg_name = &serv_addr;
msgheader.msg_namelen = sizeof(serv_addr);
msgheader.msg_iov = &iov;
msgheader.msg_iovlen = 3;
msgheader.msg_control = &mhead;
msgheader.msg_controllen = 0;
msgheader.msg_flags = 0;

if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    /* error */
}
else
{
    msg = recvmsg(sock, &msgheader, 0);
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[recvfrom](#)

[send](#)

[sendto](#)

[socket](#)

[select](#)

[sendmsg](#)

[shutdown](#)

[<sys/socket.h>](#)

[Nucleus POSIX Networking Library APIs](#)

select

Allowed from: POSIX Thread and signal handler

This function examines the file descriptor sets with addresses passed in the readfds, writefds, and errorfds parameters to determine whether some of their descriptors are ready for reading, writing, or have an exceptional condition pending.

Usage

```
int select(int nfd,
           fd_set *restrict readfds,
           fd_set *restrict writefds,
           fd_set *restrict errorfds,
           struct timeval *restrict timeout)
```

Arguments

- **nfd**
The nfd argument specifies the range of descriptors to be tested. The first nfd descriptors are checked in each set; that is, the descriptors from zero through nfd-1 in the descriptor sets are examined.
- **readfds**
If the readfds argument is not a null pointer, it points to an object of type fd_set that for input specifies the file descriptors to be checked for being ready to read, and for output indicates which file descriptors are ready to read.
- **writefds**
If the writefds argument is not a null pointer, it points to an object of type fd_set that for input specifies the file descriptors to be checked for being ready to write, and for output indicates which file descriptors are ready to write.
- **errorfds**
If the errorfds argument is not a null pointer, it points to an object of type fd_set that for input specifies the file descriptors to be checked for pending error conditions, and for output indicates which file descriptors have error conditions pending.
- **timeout**
Time to wait for this function.

Return Value

- **int**
Upon successful completion, this function returns the total number of bits set in the bit masks.
- **POSIX_ERROR**
If the call fails, it returns POSIX_ERROR and sets errno to indicate the error.

Example

```
#include "services/sys/socket.h"
#include "services/sys/select.h"

fd_set fdset;
int sock;
int sts;
struct sockaddr_in addr;
int maxsock;
int clntsock = -1;
struct timeval tv;

addr.sin_family      = AF_INET;
addr.sin_port        = htons(8888);
addr.sin_addr.s_addr = INADDR_ANY;
sock                 = socket(AF_INET, SOCK_STREAM, 0);
bind(sock, (struct sockaddr*)&addr, sizeof(addr));
listen(sock, 9);

while(1)
{
    FD_ZERO(&fdset);
    FD_PSET(sock, &fdset);
    maxsock = sock;
    if (clntsock >= 0)
    {
        FD_PSET(clntsock, &fdset);
        if (clntsock > maxsock)
            maxsock = clntsock;
    }

    tv.tv_sec  = 10;
    tv.tv_usec = 0;

    rc = select(maxsock+1, &fdset, NULL, NULL, &tv);

    ...
}
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[recvfrom](#)

[send](#)

[sendto](#)

[socket](#)

[<sys/time.h>](#)

[recv](#)

[select](#)

[sendmsg](#)

[shutdown](#)

[<sys/select.h>](#)

[Nucleus POSIX Networking Library APIs](#)

send

Allowed from: POSIX Thread and signal handler

This function initiates the transmission of a message from the specified socket to its peer. The send() function sends a message only when the socket is connected (including when the peer of a connectionless socket has been set via [connect](#)).

Usage

```
ssize_t send (int          socket,  
              const void *buffer,  
              size_t      length,  
              int          flags)
```

Arguments

- **socket**
Specifies the file descriptor associated with the socket.
- **buffer**
Points to the buffer containing the message to send.
- **length**
Specifies the length of the message in bytes.
- **flags**
Specifies the type of message transmission. Refer to [Table 5-5](#) for more information.

Table 5-5. send Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:
 - EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.
 - EBADF**

The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EDESTADDRREQ

The socket is not connection-mode and no peer address is set.

EMSGSIZE

The message is too large to be sent all at once, as the socket requires.

ENOTCONN

The socket is not connected or otherwise has not had the peer pre-specified.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The socket argument is associated with a socket that does not support one or more of the values set in flags.

EINTR

A signal interrupted send() before any data was transmitted.

EINVAL

A NULL address value or bad parameter value.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

ssize_t msg;
int sock;
struct sockaddr_in cli_addr;
char buffer[2500];
memset(buffer, 'A', 2500);

sock = socket(AF_INET, SOCK_STREAM, 0);
cli_addr.sin_family = AF_INET;
cli_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
cli_addr.sin_port = htons(8888);

if (connect(sock, (struct sockaddr *) &cli_addr,
(socklen_t) sizeof(cli_addr)) < 0)
{
    /* error */
}
```

```

else
{
    /* success */
    msg = send(sock,buffer,sizeof(buffer),0);
    sleep(1);
}

```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001 [connect](#)

[getsockopt](#) [recv](#)

[recvfrom](#) [recvmsg](#)

[select](#) [sendmsg](#)

[sendto](#) [setsockopt](#)

[shutdown](#) [socket](#)

[<sys/socket.h>](#) [Nucleus POSIX Networking Library APIs](#)

sendmsg

Allowed from: POSIX Thread and signal handler

This function sends a message through a connection-mode or connectionless-mode socket.

Usage

```
ssize_t sendmsg (int          socket,  
                 struct msghdr *message,  
                 int          flags)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **message**
Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The msg_flags member is ignored.
- **flags**
Specifies the type of message transmission. Refer to [Table 5-6](#) for more information.

Table 5-6. sendmsg Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.

Return Value

- **Number of byte**
The number of bytes sent.
- **POSIX_ERROR**
Indicates that the service fails and sets errno to the corresponding error code:
 - EAFNOSUPPORT**
Addresses in the specified address family cannot be used with this socket.
 - EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and the requested operation will block.
 - EBADF**
The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EINTR

A signal interrupted [sendto](#) before any data was transmitted.

EMSGSIZE

The message is too large to be sent all at once, as the socket requires.

ENOTCONN

The socket is not connected or otherwise has not had the peer pre-specified.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The socket argument is associated with a socket that does not support one or more of the values set in flags.

EINVAL

NULL address value or bad parameter value.

EDESTADDRREQ

The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

ENOMEM

Insufficient memory was available to fulfill the request.

EMSGSIZE

The message is too large to be sent all at once, as the socket requires.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

Description

If the socket is connectionless-mode, the message is sent to the address specified by `msghdr`. If the socket is connection-mode, the destination address in `msghdr` will be ignored.

Nucleus POSIX currently does not support ancillary data. However, Nucleus NET supports Ipv6 ancillary data for TCP and IP RAW connections. Refer to the [NET](#) chapter in the *Nucleus Networking Guide* for more information on ancillary data and its options (extended discussion section).

Example

```
int sock;  
#include "services/inet/in.h"  
#include "services/arpa/inet.h"
```

```
#include "services/sys/socket.h"

struct sockaddr_in cli_addr;
struct iovec iov[3];
struct msghdr msgheader;

char buf3[10] = "header1234";
char buf4[10] = "middlepart";
char buf5[10] = "tailpart12";

/* Use Nucleus NET ancillary data type definition */
cmsghdr mhead;

sock = socket(AF_INET, SOCK_DGRAM, 0);

cli_addr.sin_family      = AF_INET;
cli_addr.sin_addr.s_addr = inet\_addr("127.0.0.1");
cli_addr.sin_port        = htons(8888);

/* Specify the components of the message in an "iovec". */
iov[0].iov_base = &buf3;
iov[0].iov_len  = sizeof(buf3);
iov[1].iov_base = &buf4;
iov[1].iov_len  = sizeof(buf4);
iov[2].iov_base = &buf5;
iov[2].iov_len  = sizeof(buf5);

msgheader.msg_name      = &cli_addr;
msgheader.msg_namelen   = sizeof(cli_addr);
msgheader.msg_iov       = &iov;
msgheader.msg_iovlen    = 3;
msgheader.msg_control    = &mhead;
msgheader.msg_controllen = 0;
msgheader.msg_flags     = 0;
msg                     = sendmsg(sock, &msgheader, 0);

sleep(1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getsockopt
recv	recvfrom
recvmsg	select
send	sendto
setsockopt	shutdown
socket	<sys/socket.h>

[Nucleus POSIX Networking Library APIs](#)

sendto

Allowed from: POSIX Thread and signal handler

This function sends a message through a connection-mode or connectionless-mode socket.

Usage

```
ssize_t sendto (int                socket,  
                const void         *message,  
                size_t             length,  
                int                flags,  
                const struct sockaddr *dest_addr,  
                socklen_t          dest_len)
```

Arguments

- **socket**
Specifies the socket file descriptor.
- **message**
Points to a buffer containing the message to be sent.
- **length**
Specifies the size of the message in bytes.
- **flags**
Specifies the type of message transmission. Refer to [Table 5-7](#) for more information.

Table 5-7. sendto Flags

Flags	Nucleus Support	Meaning
0	SUPPORTED	Flag is not being used.
MSG_EOR	NOT SUPPORTED	Terminates a record.
MSG_OOB	NOT SUPPORTED	Sends out-of-band data on sockets that support out-of-band communications.

- **dest_addr**
Points to a sockaddr structure containing the destination address. The length and format of the address depend on the address family of the socket.
- **dest_len**
Specifies the length of the sockaddr structure pointed to by the dest_addr argument.

Return Value

- Number of bytes
The number of bytes sent.
- POSIX_ERROR

Indicates that the service fails and sets `errno` to the corresponding error code:

EAFNOSUPPORT

Addresses in the specified address family cannot be used with this socket.

EWOULDBLOCK

The socket's file descriptor is marked `O_NONBLOCK` and the requested operation will block.

EBADF

The socket argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EINTR

A signal interrupted `sendto()` before any data was transmitted.

ENOTCONN

The socket is not connected or otherwise has not had the peer pre-specified.

ENOTSOCK

The socket argument does not refer to a socket.

EOPNOTSUPP

The socket argument is associated with a socket that does not support one or more of the values set in flags.

EINVAL

A NULL address value or bad parameter value.

EDESTADDRREQ

The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

ENOMEM

Insufficient memory was available to fulfill the request.

Description

If the socket is connectionless-mode, the message is sent to the address specified by `dest_addr`.
If the socket is connection-mode, `dest_addr` is ignored.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

int sock;
int status;
struct sockaddr_in cli_addr;
char buffer[2500];

memset(buffer, 'A', 2500);

sock = socket(AF_INET, SOCK_DGRAM, 0);

cli_addr.sin_family      = AF_INET;
cli_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
cli_addr.sin_port        = htons(8888);

/* Sending message through connectionless socket*/
msg = sendto(sock, buffer, sizeof(buffer), 0,
             (struct sockaddr*)&cli_addr, sizeof(cli_addr));

sleep(1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getsockopt
recv	recvfrom
recvmsg	select
send	sendmsg
setsockopt	shutdown
socket	<sys/socket.h>
Nucleus POSIX Networking Library APIs	

sethostent

Allowed from: POSIX Thread and signal handler

This function opens a connection to the database and sets the next entry for retrieval to the first entry in the database.

Usage

```
void sethostent (int stayopen)
```

Arguments

- stayopen

If the stayopen argument is non-zero, the connection cannot be closed by a call to [gethostent](#), [gethostbyname](#), or [gethostbyaddr](#), and the implementation may maintain an open file descriptor.

Example

```
#include "services/netdb.h"

/* function opens a connection to the database and set the next entry for
   retrieval to the first entry in the database. */

sethostent(1);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[gethostbyaddr](#)

<netdb.h>

[Nucleus POSIX Networking Library APIs](#)

setsockopt

Allowed from: POSIX Thread and signal handler

This function sets the option specified by the `option_name` argument, at the protocol level specified by the `level` argument, to the value pointed to by the `option_value` argument for the socket associated with the file descriptor specified by the `socket` argument.

Usage

```
int setsockopt (int      socket,
               int      level,
               int      option_name,
               const void *option_value,
               socklen_t option_len)
```

Arguments

- `socket`
Specifies the socket file descriptor.
- `level`
Specifies the protocol level at which the option resides. Valid levels are `SOL_SOCKET`, `IPPROTO_IP`, `IPPROTO_IPV6` and `IPPROTO_TCP`. [Table 5-8](#) shows the list of levels and the associated `option_name` arguments that are currently supported:

Table 5-8. setsockopt Levels

Level	Family	Meaning
SQL_SOCKET Level		
SO_BROADCAST	IPv4	Set the broadcast status of a socket. When a socket is created the ability to send broadcasts is enabled by default.
SO_LINGER	IPv4/IPv6	Set the linger option.
SO_REUSEADDR	IPv4/IPv6	Set the option to reuse a port for a unique IP address.
IPPROTO_TCP Level		
SO_KEEPALIVE	IPv4/IPv6	Enable TCP Keep-Alive on this socket.

Table 5-8. setsockopt Levels (cont.)

Level	Family	Meaning
TCP_NODELAY	IPv4/IPv6	TCP sockets, by default, utilize the Nagle Algorithm. The Nagle Algorithm attempts to merge contiguous small TCP packets into a single large TCP packet. It does this by delaying the transmission of small packets. This better utilizes the network bandwidth. However, it is often desirable for data to be transmitted immediately. The TCP_NODELAY option can be used to control the disposition of the Nagle Algorithm. A value of 1 will eliminate the delay, for example, turn the Nagle Algorithm off. A value of 0 will enable the delay, for example, turn the Nagle Algorithm on.
IPPROTO_IP Level		
IP_ADD_MEMBERSHIP	IPv4	A multicast group. Note that setting this socket option has been deprecated in favor of the NU_IP_Multicast_Listen routine.
IP_DROP_MEMBERSHIP	IPv4	Leave a multicast group. Note that setting this socket option has been deprecated in favor of the routine NU_IP_Multicast_Listen .
IP_HDRINCL	IPv4	This option, when set, specifies that the application layer is responsible for filling in the IP header. When clear the application will only pass data to the stack, which will then append an IP header. This option is only valid when using the raw IP services.
IP_MULTICAST_TTL	IPv4	Change the default TTL (Time To Live) for multicast packets sent on this socket. The default TTL is 1. This keeps routers from forwarding the multicast datagrams beyond the local network.
IP_MULTICAST_IF	IPv4	Change the interface to be used for sending multicast datagrams. The IP address of the interface is set in the optval parameter as a 4 byte array. By default the stack searches the routing table for the interface to use.

Table 5-8. setsockopt Levels (cont.)

Level	Family	Meaning
IP_BROADCAST_IF	IPv4	Change the interface to be used for sending broadcast datagrams. The IP address of the interface is set in the optval parameter as a 4 byte array. By default, the primary interface is used.
IP_RECVIFADDR	IPv4	Enable the receive interface address option. This option enables the storing of the receive interface upon calls to NU_Recv_From . After a call to NU_Recv_From() a call to NU_Recv_IF_Addr can be made in order to obtain the IP address of the interface which received the datagram. This is useful when receiving multicast and broadcast packets that do not have a specific destination IP address. Zero disables and non-zero enables this option.
IP_TTL	IPv4	Sets the Time To Live to be used in the header of outgoing IP packets using this socket.
IP_TOS	IPv4	Retrieve the Type of Service for the specified socket. ToS is the historic name for this parameter but is really used as Diff Service Code Point (DSCP)
IPPROTO_IPV6 Level		
IPV6_CHECKSUM	IPv6	Sets the socket option for enabling or disabling the socket option for the IP layer to compute and store the checksum for outgoing RAW IPv6 packets and verify the checksum for incoming RAW IPv6 packets.
IPV6_DSTOPTS	IPv6	Sets the Destination Options extension header Sticky Option for this socket.
IPV6_DSTOPTS	IPv6	The Destination Options set will be used in all packets sent on this socket. This StickyOption value can be overridden by a call to NU_Sendmsg specifying a different Destination Options extension header as ancillary data.
IPV6_HOPOPTS	IPv6	Sets the Hop-By-Hop Options extension header Sticky Option for this socket. The Hop-By-Hop Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Hop-By-Hop Options extension header as ancillary data.

Table 5-8. setsockopt Levels (cont.)

Level	Family	Meaning
IPV6_JOIN_GROUP	IPv6	Joins a multicast group on the specified interface.
IPV6_LEAVE_GROUP	IPv6	Leaves a multicast group on the specified interface.
IPV6_MULTICAST_HOPS	IPv6	Sets the hop limit to use in multicast packets transmitted using this socket.
IPV6_MULTICAST_IF	IPv6	Sets the interface to use for multicast packets transmitted using this socket.
IPV6_NEXTHOP	IPv6	Sets the Sticky Option for the Next-Hop to be used with all packets sent from this socket. Note that this Sticky Option can be overridden on a per-packet basis with a call to <code>NU_Sendmsg</code> that specifies a different Next-Hop as ancillary data.
IPV6_PKTINFO	IPv6	Sets the source address and/or outgoing interface index Sticky Option on the socket.
IPV6_RECVDSTOPTS	IPv6	Enables or disables the value of the socket option to receive the Destination Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPLIMIT	IPv6	IPv6 Sets the socket option to return the hop limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPOPTS	IPv6	Enables or disables the value of the socket option to receive the Hop-By-Hop Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVPKTINFO	IPv6	Retrieves the value of the socket option to return the receive interface and destination address of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

Table 5-8. setsockopt Levels (cont.)

Level	Family	Meaning
IPV6_RECVRTHDR	IPv6	Enables or disables the value of the socket option to receive the Routing Header of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVTCLASS	IPv6	Sets the socket option to return the Traffic Class of the most recently received packet on this socket. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RTHDR	IPv6	Sets the Routing Header Sticky Option for this socket. The Routing Header set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to <code>NU_Sendmsg</code> specifying a different Routing Header as ancillary data.
IPV6_RTHDRDSTOPTS	IPv6	Sets the Destination Options extension header Sticky Option for the Destination Options to precede the Routing Header for this socket. The Destination Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to <code>NU_Sendmsg</code> specifying a different Destination Options extension header as ancillary data.
IPV6_TCLASS	IPv6	Sets the Traffic Class Sticky Option on the socket. The Traffic Class set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to <code>NU_Sendmsg</code> specifying a different Traffic Class as ancillary data.
IPV6_UNICAST_HOPS	IPv6	IPv6Controls the hop limit used in outgoing unicast IPV6 packets.
IPV6_V6ONLY	IPv6	Sets the socket option for a socket of family type <code>NU_FAMILY_IP6</code> to be used for IPv6 communications only. By setting this socket option, the socket can be used to send and receive IPv6 packets only.
Type	Level	Support
SO_DEBUG	N/A	NOT SUPPORTED

Table 5-8. setsockopt Levels (cont.)

Level	Family	Meaning
SO_OOINLINE	N/A	NOT SUPPORTED
SO_SNDBUF	N/A	NOT SUPPORTED
SO_RCVBUF	N/A	NOT SUPPORTED
SO_DONTROUTE	N/A	NOT SUPPORTED
SO_RCVLOWAT	N/A	NOT SUPPORTED
SO_RCVTIMEO	N/A	NOT SUPPORTED
SO_SNDLOWAT	N/A	NOT SUPPORTED
SO_SNDTIMEO	N/A	NOT SUPPORTED

- **option_name**
Specifies a single option to be retrieved. This can be set to any of the following values:

SO_BROADCAST	SO_LINGER	IP_ADD_MEMBERSHIP
IP_DROP_MEMBERSHIP	IP_MULTICAST_TTL	IP_MULTICAST_IF
IP_BROADCAST_IF	IP_RECVIFADDR	IP_HDRINCL
IP_TOS	IP_TTL	TCP_NODELAY
SO_KEEPALIVE	SO_REUSEADDR	IPV6_CHECKSUM
IPV6_DSTOPTS	IPV6_HOPOPTS	IPV6_JOIN_GROUP
IPV6_LEAVE_GROUP	IPV6_MULTICAST_HOPS	IPV6_MULTICAST_IF
IPV6_NEXTHOP	IPV6_PKTINFO	IPV6_RECVDSTOPTS
IPV6_RECVHOPLIMIT	IPV6_RECVHOPOPTS	IPV6_RECVPKTINFO
IPV6_RECVRTHDR	IPV6_RECVTCLASS	IPV6_RTHDR
IPV6_RTHDRDSTOPTS	IPV6_TCLASS	IPV6_UNICAST_HOPS
IPV6_V6ONLY		

- **option_value**
The option value of the option length.
- **option_len**
The length of the option_value in bytes.

Note



Refer to the [NET](#) chapter in the *Nucleus Networking Guide* for the option_value data structure that is outside IEEE POSIX specification. The setsockopt API can be used to set option values based on the level specified within Nucleus NET.

Return Value

- **POSIX_SUCCESS**

Indicates successful completion of the service.

- **POSIX_ERROR**

Indicates that the service fails and sets `errno` to the corresponding error code:

EBADF

The socket argument is not a valid file descriptor.

EINVAL

The specified option is invalid at the specified socket level, the socket has been shut down, or it is a NULL address value or bad parameter value.

EISCONN

The socket is already connected, and a specified option cannot be set while the socket is connected.

ENOPROTOOPT

The option is not supported by the protocol.

ENOTSOCK

The socket argument does not refer to a socket.

ENOMEM

There was insufficient memory available for the operation to complete.

Example

```
#include "services/inet/in.h"
#include "services/arpa/inet.h"
#include "services/sys/socket.h"

/* setting SO_REUSEADDR to true */
int setval = 1;
setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &setval, sizeof(setval));

/* setval data is different depending on the level and option name.
   Please refer to Nucleus NET for option name outside IEEE POSIX
   specification */
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001

[endprotoent](#)

[getsockopt](#)

[<netinet/in.h>](#)

[setsockopt](#)

[shutdown](#)

[socket](#)

[<sys/socket.h>](#)

[Nucleus POSIX Networking Library APIs](#)

shutdown

Allowed from: POSIX Thread and signal handler

This function shuts down all or part of a full-duplex connection on the file descriptor socket.

Usage

```
int shutdown (int socket,  
             int flag)
```

Arguments

- **socket**
Specifies the file descriptor of the socket.
- **flag**
Specifies the type of shutdown. Refer to [Table 5-9](#) for more information.

Table 5-9. shutdown Flags

Flags	Nucleus Support	Meaning
SHUT_RD	SUPPORTED	Disables further receive operations.
SHUT_WR	SUPPORTED	Disables further send operations.
SHUT_RDWR	SUPPORTED	Disables further send and receive operations.

Return Value

- **POSIX_SUCCESS**
Indicates successful completion of the service.
- **POSIX_ERROR**
Indicates that the service has failed and sets the `errno` to the corresponding error code:
 - EBADF**
The socket argument is not a valid file descriptor.
 - EINVAL**
The how argument is invalid.
 - ENOTCONN**
The socket is not connected.
 - ENOTSOCK**
The socket argument does not refer to a socket.

Example

```
#include "services/arpa/inet.h"  
#include "services/sys/socket.h"
```

```
int sock;

sock = socket(AF_INET, SOCK_STREAM, 0);

/* ... Do all the necessary setup */

/* shut down socket send */
shutdown(sock, SHUT_WR);

/* or shut down socket receive */
shutdown(sock, SHUT_RD);
```

Related Topics

Base Definitions volume of IEEE Std 1003.1-2001	getsockopt
read	recv
recvfrom	recvmsg
select	send
sendto	setsockopt
socket	<sys/socket.h>
write	Nucleus POSIX Networking Library APIs

socket

Allowed from: POSIX Thread and signal handler

This function creates an unbound socket in a communications domain, and returns a file descriptor that can be used in later function calls that operate on sockets. This function supports TCP, UDP and IPRAW socket types.

Usage

```
int socket (int domain,
            int type,
            int protocol)
```

Arguments

- **domain**
Specifies the communications domain in which a socket is to be created.
- **type**
Specifies the type of socket to be created. Refer to [Table 5-10](#) for more information.

Table 5-10. Socket Types

Type	Nucleus Support	Meaning
SOCK_STREAM	SUPPORTED	Byte-stream socket.
SOCK_DGRAM	SUPPORTED	Datagram socket.
SOCK_RAW	SUPPORTED	Raw Protocol Interface.
SOCK_SEQPACKET	NOT SUPPORTED	Sequenced-packet socket.

- **protocol**
Specifies a particular protocol to be used with the socket. Specifying a protocol of 0 causes `socket()` to use an unspecified default protocol appropriate for the requested socket type. Refer to [Table 5-11](#) for more information.

Table 5-11. Protocol Types

SOCK_STREAM or SOCK_DGRAM Type	Meaning
0	The protocol field is not used.
SOCK_RAW Type	Meaning
IPPROTO_HELLO	HELLO routing protocol.*
IPPROTO OSPF	OSPF routing protocol.*
IPPROTO_RAW	Raw IP protocol.*
0	Raw IP wildcard protocol.*

Refer to the [NET](#) chapter in the *Nucleus Networking Guide* for further details.

Note

(SOCK_RAW) If the recipient is waiting for a data gram of zero protocol, then the received datagram's protocol field can be any nonzero protocol to be accepted. Notice that if a Raw IP socket is created with a protocol of 0, and the socket is not bound to any local address, then the socket will receive a copy of every incoming raw datagram

Return Value

- file descriptor

New socket file descriptor.

- POSIX_ERROR

If the call fails, it returns -1 and sets errno to the corresponding error:

EAGAIN

The implementation does not support the specified address family.

EAFNOSUPPORT

No more file descriptors are available for this process.

EMFILE

No more file descriptors are available for the system.

EPROTONOSUPPORT

The protocol is not supported by the address family, or the protocol is not supported by the implementation.

EPROTOTYPE

The socket type is not supported by the protocol.

ENOMEM

Insufficient memory was available to fulfill the request.

Example

```
#include "services/sys/socket.h"
int sock;
sock = socket(AF_INET, SOCK_STREAM, 0);
```

Related Topics

[accept](#)

Base Definitions volume of IEEE Std 1003.1-2001

[bind](#)

[connect](#)

[getsockname](#)

[getsockopt](#)

[listen](#)

[recv](#)

[recvfrom](#)

[send](#)

[setsockopt](#)

[<sys/socket.h>](#)

[recvmsg](#)

[sendmsg](#)

[shutdown](#)

[Nucleus POSIX Networking Library APIs](#)

Chapter 6

Nucleus POSIX Asynchronous I/O Library APIs

This chapter provides a detailed reference of all the Nucleus POSIX Asynchronous I/O Library (POSIX AIO) APIs.

- [aio_cancel](#)
- [aio_disable](#)
- [aio_enable](#)
- [aio_error](#)
- [aio_read](#)
- [aio_return](#)
- [aio_suspend](#)
- [aio_write](#)
- [lio_listio](#)

aio_cancel

Allowed from: POSIX Thread and signal handler

This service cancels one or more asynchronous I/O requests currently outstanding against the descriptor `filedes`.

Usage

```
int aio_cancel (int      filedes,
                struct aiocb *aiocbp)
```

Arguments

- `filedes`
File descriptor.
- `aiocbp`
Pointer to the asynchronous I/O control block for a particular request to be canceled.

Return Value

- `AIO_CANCELED`
All the operations are cancelled.
- `AIO_NOTCANCELED`
At least one of the operations is not cancelled because it is in progress.
- `AIO_ALLDONE`
If all the operations have already completed.
- `POSIX_ERROR`
Indicates that the service has failed and sets `errno` to the corresponding error code:
 - `EBADF`
File descriptor is not a valid descriptor.

Example

```
#include "services/aio.h"

struct aiocb aiocbp;
int ret;
int fd;

...

ret = aio_cancel(fd, &aiocbp);
```

Note



In our implementation `aio_cancel()` can only cancel the specified request. It does not have the capability to cancel all the requests pending against the given descriptor.

Related Topics

[aio_read](#)

[<aio.h>](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

[aio_write](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

aio_disable

Allowed from: POSIX Thread and signal handler

This service disables asynchronous I/O capability for the device pointed to by devname.

Usage

```
int aio_disable (char *devname)
```

Arguments

- devname
Pointer to the device name.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service has failed and sets errno to the corresponding error code:
ENOENT
No device was found corresponding to the given devname.

Example

```
#include "services/aio.h"

int ret;

...

/* Disables Asynchronous I/O capability for the console device. */
ret = aio_disable("/dev/console");
```

Related Topics

[aio_enable](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

aio_enable

Allowed from: POSIX Thread and signal handler

This service enables asynchronous I/O capability for the device pointed to by devname.

Usage

```
int aio_enable (char *devname,  
               int  priority)
```

Arguments

- devname
Pointer to the device name.
- priority
Priority of the asynchronous I/O handling thread to be created.

Return Value

- POSIX_SUCCESS
Indicates successful completion of the service.
- POSIX_ERROR
Indicates that the service has failed and sets errno to the corresponding error code:
 - ENOENT
No device was found corresponding to the given evname.
 - EAGAIN
System lacked the necessary resources.
 - ENFILE
Too many message queue descriptors are currently in use.
 - ENOSPC
There is insufficient space available for the creation of new message queue.

Example

```
#include "services/aio.h"  
  
int ret;  
  
...  
  
/* Enables Asynchronous I/O capability for the console device and set  
   priority equal to 130 for the AIO thread to be created */  
ret = aio_enable("/dev/console",130);
```

Related Topics

[aio_disable](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

aio_error

Allowed from: POSIX Thread and signal handler

This service returns the error status associated with the specified aiocb structure.

Usage

```
int aio_error (const struct aiocb *aiocbp)
```

Arguments

- **aiocbp**
Pointer to the asynchronous I/O control block against which we are going to check the error status.

Return Value

- **POSIX_SUCCESS**
If the asynchronous operation is completed successfully.
- Error returned by [read](#) or [write](#)
If the asynchronous operation has been completed unsuccessfully.
- **EINPROGRESS**
If the asynchronous I/O Operation has not yet completed.
- **POSIX_ERROR**
Indicates that the service has failed and sets `errno` to the corresponding error code:
EINVAL
The `aiocbp` argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

Description

The error status for an asynchronous I/O operation is the `errno` value that is set by the corresponding [read](#) or [write](#) operation. If the operation has not yet completed, then the error status is equal to **EINPROGRESS**.

Example

```
#include "services/aio.h"

struct aiocb aiocbp;
int ret_status;

...

ret_status = aio_error(&aiocbp);
```

Related Topics

[aio_cancel](#)

[aio_return](#)

[<aio.h>](#)

[close](#)

[lseek](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

[aio_read](#)

[aio_write](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[lio_listio](#)

[read](#)

aio_read

Allowed from: POSIX Thread and signal handler

This service reads `aioctx->aio_nbytes` from the file associated with `aioctx->aio_fildes` into the buffer pointed to by `aioctx->aio_buf`. The function call returns when the read request is initiated or queued to the file or device (even when the data cannot be delivered immediately).

Usage

```
int aio_read (struct aiocb *aioctx)
```

Arguments

- `aioctx`
Pointer to asynchronous I/O control block.

Return Value

- `POSIX_SUCCESS`
If the I/O operation is successfully queued.
- `POSIX_ERROR`
If the I/O operation cannot be queued. It then sets `errno` to the corresponding error code:
`EAGAIN`
The requested asynchronous I/O operation was not queued due to system resource limitations.

Example

```
#include "services/aio.h"

struct aiocb aioctx;
int ret;

...

ret = aio_read(&aioctx);
```

Related Topics

[aio_cancel](#)

[aio_return](#)

[<aio.h>](#)

[lio_listio](#)

[lseek](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

[aio_error](#)

[aio_write](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[close](#)

[read](#)

aio_return

Allowed from: POSIX Thread and signal handler

This function returns the return status associated with the aiocb structure referenced by the aiocbp argument.

Usage

```
ssize_t aio_return (struct aiocb *aiocbp)
```

Arguments

- **aiocbp**
Pointer to asynchronous I/O control block.

Return Value

- **ssize_t**
The status associated with the aiocb structure.
- **POSIX_ERROR**
If the I/O operation cannot be queued. It then sets errno to the corresponding error code:
EINVAL
The aiocbp argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

Example

```
#include "services/aio.h"

struct aiocb aiocbp;
int ret;

...

ret = aio_return(&aiocbp);
```

Related Topics

[aio_cancel](#)

[aio_read](#)

[<aio.h>](#)

[close](#)

[lseek](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

[aio_error](#)

[aio_write](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[lio_listio](#)

[read](#)

aio_suspend

Allowed from: POSIX Thread and signal handler

This service suspends the calling thread until at least one of the asynchronous I/O operations referenced by the list argument has completed. This call cannot be interrupted by the delivery of a signal.

Usage

```
int aio_suspend (const struct aiocb* const list[],
                 int nent,
                 const struct timespec *timeout)
```

Arguments

- **list**
List of the pointer to asynchronous I/O control blocks.
- **nent**
Number of entries in the list.
- **timeout**
Timeout value for the suspension.

Return Value

- **POSIX_SUCCESS**
Returns zero after one or more asynchronous I/O operations completed.
- **POSIX_ERROR**
If no asynchronous I/O indicated in the list have completed. It then sets `errno` to the corresponding error code:
 - EAGAIN**
The requested asynchronous I/O operation was not queued due to system resource limitations.

Example

```
#include "services/aio.h"

const struct aiocb aiocbp const list[];
int ret;

...

ret = aio_suspend(&list,10,0);
```

Related Topics

[aio_read](#)

[aio_write](#)

<aio.h>

Base Definitions volume of IEEE Std 1003.1-2001

[lio_listio](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

aio_write

Allowed from: POSIX Thread and signal handler

This service writes a `aioebp->aio_nbytes` to the file associated with `aioebp->aio_fildes` from the buffer pointed to by `aioebp->aio_buf`. The function returns when the write request has been initiated or, at a minimum, queued to the file or device.

Usage

```
int aio_write (struct aiocb* aioebp)
```

Arguments

- `aioebp`
Pointer to asynchronous I/O control block.

Return Value

- `POSIX_SUCCESS`
If the I/O operation is successfully queued.
- `POSIX_ERROR`
If the I/O operation cannot be queued. It then sets `errno` to the corresponding error code:
`EAGAIN`
The requested asynchronous I/O operation was not queued due to system resource limitations.

Example

```
#include "services/aio.h"

struct aiocb aioebp;
int ret;

...

ret = aio_write(&aioebp);
```

Related Topics

[aio_cancel](#)

[aio_read](#)

[<aio.h>](#)

[Base Definitions volume of IEEE Std 1003.1-2001](#)

[lseek](#)

[Nucleus POSIX Asynchronous I/O Library APIs](#)

[aio_error](#)

[aio_return](#)

[close](#)

[lio_listio](#)

[write](#)

lio_listio

Allowed from: POSIX Thread and signal handler

This service initiates a list of I/O requests.

Usage

```
int lio_listio (int          mode,
                struct aiocb *const list[],
                int          nent,
                struct sigevent *sig)
```

Arguments

- mode

Determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. This can be set to one of the following values:

LIO_WAIT

Indicating that the calling thread is to suspend until the `lio_listio()` operation completes.

LIO_NOWAIT

Indicating that the calling thread is to continue operation while the `lio_listio()` operation is being performed and no notification when the operation is complete.

- list

Pointer to the I/O list.

- nent

Number of entries in the list.

- sig

Pointer Signal event structure if not NULL causes the asynchronous notification to be sent when all the I/O operations complete.

Return Value

- POSIX_SUCCESS

If mode is `LIO_NOWAIT`, the I/O operation is successfully queued.

If mode is `LIO_WAIT`, then the I/O operation has been completed successfully.

- POSIX_ERROR

If any asynchronous I/O indicated in the list not have completed or cannot be queued. It then sets `errno` to the corresponding error code:

ENOTSUP

If the mode is `LIO_NOWAIT` and the signal is not NULL because it is not supported.

EINVAL

The mode argument is not a proper value, or the value of nent is greater than AIO_LISTIO_MAX.

Description

An asynchronous notification is not supported upon the completion of all the asynchronous requests as specified in the lio_listio. In this implementation, the call cannot be interrupted by the delivery of a signal.

Example

```
#include "services/aio.h"

struct aiocb aiocbp const list[];
int ret;

...

ret = lio_listio(LIO_WAIT, &list, 10, NULL);
```

Related Topics[aio_cancel](#)[aio_error](#)[aio_read](#)[aio_return](#)[aio_write](#)[<aio.h>](#)[Base Definitions volume of IEEE Std 1003.1-2001](#)[close](#)[lseek](#)[read](#)[Nucleus POSIX Asynchronous I/O Library APIs](#)

Embedded Software and Hardware License Agreement

The latest version of the Embedded Software and Hardware License Agreement is available on-line at:
www.mentor.com/eshla

IMPORTANT INFORMATION

USE OF ALL PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Products received electronically, certify destruction of Products and all accompanying items within five days after receipt of such Products and receive a full refund of any license fee paid.

1. **Definitions.** As used in this Agreement and any applicable quotation, supplement, attachment and/or addendum ("Addenda"), these terms shall have the following meanings:
 - 1.1. "Customer's Product" means Customer's end-user product identified by a unique SKU (including any Related SKUs) in an applicable Addenda that is developed, manufactured, branded and shipped solely by Customer or an authorized manufacturer or subcontractor on behalf of Customer to end-users or consumers;
 - 1.2. "Developer" means a unique user, as identified by a unique user identification number, with access to Embedded Software at an authorized Development Location. A unique user is an individual who works directly with the embedded software in source code form, or creates, modifies or compiles software that ultimately links to the Embedded Software in Object Code form and is embedded into Customer's Product at the point of manufacture;
 - 1.3. "Development Location" means the location where Products may be used as authorized in the applicable Addenda;
 - 1.4. "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product;
 - 1.5. "Embedded Software" means Software that is embeddable;
 - 1.6. "End-User" means Customer's customer;
 - 1.7. "Executable Code" means a compiled program translated into a machine-readable format that can be loaded into memory and run by a certain processor;
 - 1.8. "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation;
 - 1.9. "Linkable Object Code" or "Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine-readable format;
 - 1.10. "Mentor Embedded Linux" or "MEL" means Mentor Graphics' tools, source code, and recipes for building Linux systems;
 - 1.11. "Open Source Software" or "OSS" means software subject to an open source license which requires as a condition for redistribution of such software, including modifications thereto, that the: (i) redistribution be in source code form or be made available in source code form; (ii) redistributed software be licensed to allow the making of derivative works; or (iii) redistribution be at no charge;
 - 1.12. "Processor" means the specific microprocessor to be used with Software and implemented in Customer's Product;
 - 1.13. "Products" means Software, Term-Licensed Products and/or Hardware;
 - 1.14. "Proprietary Components" means the components of the Products that are owned and/or licensed by Mentor Graphics and are not subject to an Open Source Software license, as more fully set forth in the product documentation provided with the Products;

- 1.15. “Redistributable Components” means those components that are intended to be incorporated or linked into Customer’s Linkable Object Code developed with the Software, as more fully set forth in the documentation provided with the Products;
- 1.16. “Related SKU” means two or more Customer Products identified by logically-related SKUs, where there is no difference or change in the electrical hardware or software content between such Customer Products;
- 1.17. “Software” means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement;
- 1.18. “Source Code” means software in a form in which the program logic is readily understandable by a human being;
- 1.19. “Sourcery CodeBench Software” means Mentor Graphics’ Development Tool for C/C++ embedded application development;
- 1.20. “Sourcery VSIPL++” is Software providing C++ classes and functions for writing embedded signal processing applications designed to run on one or more processors;
- 1.21. “Stock Keeping Unit” or “SKU” is a unique number or code used to identify each distinct product, item or service available for purchase;
- 1.22. “Subsidiary” means any corporation more than 50% owned by Customer, excluding Mentor Graphics competitors. Customer agrees to fulfill the obligations of such Subsidiary in the event of default. To the extent Mentor Graphics authorizes any Subsidiary’s use of Products under this Agreement, Customer agrees to ensure such Subsidiary’s compliance with the terms of this Agreement and will be liable for any breach by a Subsidiary; and
- 1.23. “Term-Licensed Products” means Products licensed to Customer for a limited time period (“Term”).

2. Orders, Fees and Payment.

- 2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer’s appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (“Order(s)”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement and any applicable Addenda, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. All invoices will be sent electronically to Customer on the date stated on the invoice unless otherwise specified in an Addendum. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer’s sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer’s behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 2.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics’ delivery of Software by electronic means is subject to Customer’s provision of both a primary and an alternate e-mail address.

3. Grant of License.

- 3.1. The Products installed, downloaded, or otherwise acquired by Customer under this Agreement constitute or contain copyrighted, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as described in the applicable Addenda. The limited licenses granted under the applicable Addenda shall continue until the expiration date of Term-Licensed Products or termination in accordance with Section 12 below, whichever occurs first. **Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement or Addenda with Mentor Graphics for such purpose.**
- 3.2. License Type. The license type shall be identified in the applicable Addenda.
- 3.2.1. Development License: During the Term, if any, Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form by the number of Developers specified, for the Processor(s), Customer’s Product(s) and at the Development Location(s) identified in the applicable Addenda.

- 3.2.2. End-User Product License: During the Term, if any, and unless otherwise specified in the applicable Addenda, Customer may incorporate or embed an Executable Code version of the Embedded Software into the specified number of copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the applicable Addenda. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.
- 3.2.3. Internal Tool License: During the Term, if any, Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat or floating basis, as specified in the applicable Addenda, and shall not be distributed to others or delivered in Customer's Product(s) unless specifically authorized in an applicable Addenda.
- 3.2.4. Sourcery CodeBench Professional Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software (i) if the license is a node-locked license, by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, or (ii) if the license is a floating license, by the authorized number of concurrent users on one or more machines provided that only the authorized number of copies of the Software are in use at any one time, and (b) distribute the Redistributable Components of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.5. Sourcery CodeBench Standard Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.6. Sourcery CodeBench Personal Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.7. Sourcery CodeBench Academic Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software for non-commercial, academic purposes only by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.3. Mentor Graphics may from time to time, in its sole discretion, lend Products to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the Term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's internal evaluation in a non-production environment. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan Term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

4. Beta Code.

- 4.1. Portions or all of certain Products may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. Restrictions on Use.

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees, authorized manufacturers or authorized contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use.
- 5.2. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent, which shall not be unreasonably withheld, and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. Notwithstanding any provision in an OSS license agreement applicable to a component of the Sourcery CodeBench Software that permits the redistribution of such component to a third party in Source Code or binary form, Customer may not use any Mentor Graphics trademark, whether registered or unregistered, in connection with such distribution, and may not recompile the Open Source Software components with the --with-pkgversion or --with-bugurl configuration options that embed Mentor Graphics' trademarks in the resulting binary.
- 5.5. The provisions of this Section 5 shall survive the termination of this Agreement.

6. Support Services.

- 6.1. Except as described in Sections 6.2, 6.3 and 6.4 below, and unless otherwise specified in any applicable Addenda to this Agreement, to the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current End-User Software Support Terms located at <http://supportnet.mentor.com/about/legal/>.
- 6.2. To the extent Customer purchases support services for Sourcery CodeBench Software, support will be provided solely in accordance with the provisions of this Section 6.2. Mentor Graphics shall provide updates and technical support to Customer as described herein only on the condition that Customer uses the Executable Code form of the Sourcery CodeBench Software for internal use only and/or distributes the Redistributable Components in Executable Code form only (except as provided in a separate redistribution agreement with Mentor Graphics or as required by the applicable Open Source license). Any other distribution by Customer of the Sourcery CodeBench Software (or any component thereof) in any form, including distribution permitted by the applicable Open Source license, shall automatically terminate any remaining support term. Subject to the foregoing and the payment of support fees, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current Sourcery CodeBench Software Support Terms located at <http://www.mentor.com/codebench-support-legal>.
- 6.3. To the extent Customer purchases support services for Sourcery VSIPL++, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Sourcery VSIPL++ Support Terms located at <http://www.mentor.com/vsipl-support-legal>.
- 6.4. To the extent Customer purchases support services for Mentor Embedded Linux, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased

solely in accordance with Mentor Graphics' then-current Mentor Embedded Linux Support Terms located at <http://www.mentor.com/mel-support-legal>.

7. **Third Party and Open Source Software.** Products may contain Open Source Software or code distributed under a proprietary third party license agreement. Please see applicable Products documentation, including but not limited to license notice files, header files or source code for further details. Please see the applicable Open Source Software license(s) for additional rights and obligations governing your use and distribution of Open Source Software. Customer agrees that it shall not subject any Product provided by Mentor Graphics under this Agreement to any Open Source Software license that does not otherwise apply to such Product. In the event of conflict between the terms of this Agreement, any Addenda and an applicable OSS or proprietary third party agreement, the OSS or proprietary third party agreement will control solely with respect to the OSS or proprietary third party software component. The provisions of this Section 7 shall survive the termination of this Agreement.
8. **Limited Warranty.**
 - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; OR (B) PRODUCTS PROVIDED AT NO CHARGE, WHICH ARE PROVIDED "AS IS" UNLESS OTHERWISE AGREED IN WRITING.
 - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, AND EXCEPT FOR EITHER PARTY'S BREACH OF ITS CONFIDENTIALITY OBLIGATIONS, CUSTOMER'S BREACH OF LICENSING TERMS OR CUSTOMER'S OBLIGATIONS UNDER SECTION 10, IN NO EVENT SHALL: (A) EITHER PARTY OR ITS RESPECTIVE LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF SUCH PARTY OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; AND (B) EITHER PARTY OR ITS RESPECTIVE LICENSORS' LIABILITY UNDER THIS AGREEMENT, INCLUDING, FOR THE AVOIDANCE OF DOUBT, LIABILITY FOR ATTORNEYS' FEES OR COSTS, EXCEED THE GREATER OF THE FEES PAID OR OWING TO MENTOR GRAPHICS FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM OR \$500,000 (FIVE HUNDRED THOUSAND U.S. DOLLARS). NOTWITHSTANDING THE FOREGOING, IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **Hazardous Applications.**
 - 10.1. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other applications in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").
 - 10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATIONS.
 - 10.3. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING REASONABLE ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10.1.
 - 10.4. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. Infringement.

- 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
 - 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, and in addition to its obligations under Section 11.1, either (a) replace or modify the Product so that it becomes noninfringing; or (b) procure for Customer the right to continue using the Product. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of the Product and refund to Customer any purchase price or license fee(s) paid.
 - 11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics, where the Product itself is not infringing; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics, where the unmodified Product would not infringe; (c) the use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product of substantially similar functionality in accordance with Subsection 11.2(a); (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells, where the Product itself is not infringing; (f) any Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) Open Source Software, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such Open Source Software; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorneys' fees and other costs related to the action.
 - 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 9 ABOVE AND STATES: (A) THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND (B) CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
12. **Termination and Effect of Termination.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized Term.
- 12.1. Termination for Breach. This Agreement shall remain in effect until terminated in accordance with its terms. Mentor Graphics may terminate this Agreement and/or any licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products, if Customer (a) commits any material breach of any provision of this Agreement and fails to cure such breach upon 30-days prior written notice; or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination. For the avoidance of doubt, nothing in this Section 12 shall be construed to prevent Mentor Graphics from seeking immediate injunctive relief in the event of any threatened or actual breach of Customer's obligations hereunder.
 - 12.2. Effect of Termination. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination or expiration of the Term, Customer will discontinue use and/or distribution of Products, and shall return Hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form, except to the extent an Open Source Software license conflicts with this Section 12.2 and permits Customer's continued use of any Open Source Software portion or component of a Product. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as: (a) the End-User was licensed according to the terms of this Agreement, if applicable to such End-User, and (b) such End-User is not in breach of its agreement, if applicable, nor a party to Customer's breach.
13. **Export.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.
14. **U.S. Government License Rights.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

15. **Third Party Beneficiary.** For any Products licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **Review of License Usage.** Customer will monitor the access to and use of Software. With prior written notice, during Customer's normal business hours, and no more frequently than once per calendar year, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system, records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **Controlling Law, Jurisdiction and Dispute Resolution.** This Agreement shall be governed by and construed under the laws of the State of California, USA, excluding choice of law rules. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the state and federal courts of California, USA. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer or its Subsidiary in the jurisdiction where Customer's or its Subsidiary's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **Severability.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **Miscellaneous.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.