

PennOS

1.0

Generated by Doxygen 1.10.0

1 README	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
4.1 spthread_fwd_args_st Struct Reference	7
4.1.1 Field Documentation	7
4.1.1.1 actual_arg	7
4.1.1.2 actual_routine	7
4.1.1.3 child_meta	7
4.1.1.4 setup_cond	7
4.1.1.5 setup_done	8
4.1.1.6 setup_mutex	8
4.2 spthread_meta_st Struct Reference	8
4.2.1 Field Documentation	8
4.2.1.1 meta_mutex	8
4.2.1.2 state	8
4.2.1.3 suspend_set	8
4.3 spthread_signal_args_st Struct Reference	8
4.3.1 Field Documentation	9
4.3.1.1 ack	9
4.3.1.2 shutup_mutex	9
4.3.1.3 signal	9
4.4 spthread_st Struct Reference	9
4.4.1 Field Documentation	9
4.4.1.1 meta	9
4.4.1.2 thread	9
5 File Documentation	11
5.1 doc/README.md File Reference	11
5.2 src/pennfat.c File Reference	11
5.3 src/pennos.c File Reference	11
5.4 src/util/kernel.c File Reference	11
5.5 src/util/kernel.h File Reference	11
5.5.1 Function Documentation	11
5.5.1.1 k_proc_cleanup()	11
5.5.1.2 k_proc_create()	12
5.6 kernel.h	12
5.7 src/util/spthread.c File Reference	12
5.7.1 Macro Definition Documentation	13

5.7.1.1	<code>_GNU_SOURCE</code>	13
5.7.1.2	<code>_XOPEN_SOURCE</code>	13
5.7.1.3	<code>MILISEC_IN_NANO</code>	13
5.7.1.4	<code>SPTHREAD_RUNNING_STATE</code>	13
5.7.1.5	<code>SPTHREAD_SIG_CONTINUE</code>	13
5.7.1.6	<code>SPTHREAD_SIG_SUSPEND</code>	13
5.7.1.7	<code>SPTHREAD_SUSPENDED_STATE</code>	14
5.7.1.8	<code>SPTHREAD_TERMINATED_STATE</code>	14
5.7.2	Typedef Documentation	14
5.7.2.1	<code>pthread_fn</code>	14
5.7.2.2	<code>spthread_fwd_args</code>	14
5.7.2.3	<code>spthread_meta_t</code>	14
5.7.2.4	<code>spthread_signal_args</code>	14
5.7.3	Function Documentation	14
5.7.3.1	<code>spthread_cancel()</code>	14
5.7.3.2	<code>spthread_continue()</code>	14
5.7.3.3	<code>spthread_create()</code>	14
5.7.3.4	<code>spthread_exit()</code>	15
5.7.3.5	<code>spthread_join()</code>	15
5.7.3.6	<code>spthread_self()</code>	15
5.7.3.7	<code>spthread_suspend()</code>	15
5.7.3.8	<code>spthread_suspend_self()</code>	15
5.8	<code>src/util/spthread.h</code> File Reference	15
5.8.1	Macro Definition Documentation	16
5.8.1.1	<code>SIGPTH</code>	16
5.8.2	Typedef Documentation	16
5.8.2.1	<code>spthread_meta_t</code>	16
5.8.2.2	<code>spthread_t</code>	16
5.8.3	Function Documentation	16
5.8.3.1	<code>spthread_cancel()</code>	16
5.8.3.2	<code>spthread_continue()</code>	16
5.8.3.3	<code>spthread_create()</code>	16
5.8.3.4	<code>spthread_exit()</code>	17
5.8.3.5	<code>spthread_join()</code>	17
5.8.3.6	<code>spthread_self()</code>	17
5.8.3.7	<code>spthread_suspend()</code>	17
5.8.3.8	<code>spthread_suspend_self()</code>	17
5.9	<code>spthread.h</code>	17
5.10	<code>src/util/sys_call.c</code> File Reference	20
5.11	<code>src/util/sys_call.h</code> File Reference	20
5.11.1	Function Documentation	20
5.11.1.1	<code>s_exit()</code>	20

5.11.1.2 s_kill()	20
5.11.1.3 s_nice()	21
5.11.1.4 s_sleep()	21
5.11.1.5 s_spawn()	21
5.11.1.6 s_waitpid()	22
5.12 sys_call.h	22
5.13 test/sched-demo.c File Reference	22
5.13.1 Macro Definition Documentation	23
5.13.1.1 BUF_SIZE	23
5.13.1.2 NUM_THREADS	23
5.13.2 Function Documentation	23
5.13.2.1 cancel_and_join()	23
5.13.2.2 main()	23
Index	25

Chapter 1

README

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

spthread_fwd_args_st	7
spthread_meta_st	8
spthread_signal_args_st	8
spthread_st	9

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/ pennfat.c	11
src/ pennos.c	11
src/util/ kernel.c	11
src/util/ kernel.h	11
src/util/ spthread.c	12
src/util/ spthread.h	15
src/util/ sys_call.c	20
src/util/ sys_call.h	20
test/ sched-demo.c	22

Chapter 4

Data Structure Documentation

4.1 `spthread_fwd_args_st` Struct Reference

Data Fields

- `pthread_fn` `actual_routine`
- `void *` `actual_arg`
- `bool` `setup_done`
- `pthread_mutex_t` `setup_mutex`
- `pthread_cond_t` `setup_cond`
- `spthread_meta_t *` `child_meta`

4.1.1 Field Documentation

4.1.1.1 `actual_arg`

```
void* spthread_fwd_args_st::actual_arg
```

4.1.1.2 `actual_routine`

```
pthread_fn spthread_fwd_args_st::actual_routine
```

4.1.1.3 `child_meta`

```
spthread_meta_t* spthread_fwd_args_st::child_meta
```

4.1.1.4 `setup_cond`

```
pthread_cond_t spthread_fwd_args_st::setup_cond
```

4.1.1.5 setup_done

```
bool spthread_fwd_args_st::setup_done
```

4.1.1.6 setup_mutex

```
pthread_mutex_t spthread_fwd_args_st::setup_mutex
```

The documentation for this struct was generated from the following file:

- [src/util/spthread.c](#)

4.2 spthread_meta_st Struct Reference

Data Fields

- sigset_t [suspend_set](#)
- volatile sig_atomic_t [state](#)
- pthread_mutex_t [meta_mutex](#)

4.2.1 Field Documentation

4.2.1.1 meta_mutex

```
pthread_mutex_t spthread_meta_st::meta_mutex
```

4.2.1.2 state

```
volatile sig_atomic_t spthread_meta_st::state
```

4.2.1.3 suspend_set

```
sigset_t spthread_meta_st::suspend_set
```

The documentation for this struct was generated from the following file:

- [src/util/spthread.c](#)

4.3 spthread_signal_args_st Struct Reference

Data Fields

- const int [signal](#)
- volatile sig_atomic_t [ack](#)
- pthread_mutex_t [shutup_mutex](#)

4.3.1 Field Documentation

4.3.1.1 ack

```
volatile sig_atomic_t spthread_signal_args_st::ack
```

4.3.1.2 shutup_mutex

```
pthread_mutex_t spthread_signal_args_st::shutup_mutex
```

4.3.1.3 signal

```
const int spthread_signal_args_st::signal
```

The documentation for this struct was generated from the following file:

- [src/util/spthread.c](#)

4.4 spthread_st Struct Reference

```
#include <spthread.h>
```

Data Fields

- [pthread_t](#) [thread](#)
- [spthread_meta_t](#) * [meta](#)

4.4.1 Field Documentation

4.4.1.1 meta

```
spthread\_meta\_t* spthread_st::meta
```

4.4.1.2 thread

```
pthread_t spthread_st::thread
```

The documentation for this struct was generated from the following file:

- [src/util/spthread.h](#)

Chapter 5

File Documentation

5.1 doc/README.md File Reference

5.2 src/pennfat.c File Reference

5.3 src/pennos.c File Reference

5.4 src/util/kernel.c File Reference

```
#include "kernel.h"
```

5.5 src/util/kernel.h File Reference

Functions

- `pcb_t * k_proc_create (pcb_t *parent)`
Create a new child process, inheriting applicable properties from the parent.
- `void k_proc_cleanup (pcb_t *proc)`
Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.

5.5.1 Function Documentation

5.5.1.1 `k_proc_cleanup()`

```
void k_proc_cleanup (  
    pcb_t * proc )
```

Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.

5.5.1.2 k_proc_create()

```
pcb_t * k_proc_create (
    pcb_t * parent )
```

Create a new child process, inheriting applicable properties from the parent.

Returns

Reference to the child PCB.

5.6 kernel.h

[Go to the documentation of this file.](#)

```
00001
00006 pcb_t* k_proc_create(pcb_t *parent);
00007
00012 void k_proc_cleanup(pcb_t *proc);
```

5.7 src/util/spthread.c File Reference

```
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include "../spthread.h"
#include <string.h>
#include <stdio.h>
```

Data Structures

- struct [spthread_fwd_args_st](#)
- struct [spthread_signal_args_st](#)
- struct [spthread_meta_st](#)

Macros

- #define [_GNU_SOURCE](#)
- #define [_XOPEN_SOURCE](#) 700
- #define [MILISEC_IN_NANO](#) 100000
- #define [SPTHREAD_RUNNING_STATE](#) 0
- #define [SPTHREAD_SUSPENDED_STATE](#) 1
- #define [SPTHREAD_TERMINATED_STATE](#) 2
- #define [SPTHREAD_SIG_SUSPEND](#) -1
- #define [SPTHREAD_SIG_CONTINUE](#) -2

Typedefs

- typedef void [*\(* pthread_fn\)](#) (void *)
- typedef struct [spthread_fwd_args_st](#) [spthread_fwd_args](#)
- typedef struct [spthread_signal_args_st](#) [spthread_signal_args](#)
- typedef struct [spthread_meta_st](#) [spthread_meta_t](#)

Functions

- int [spthread_create](#) ([spthread_t](#) *thread, const pthread_attr_t *attr, [pthread_fn](#) start_routine, void *arg)
- int [spthread_suspend](#) ([spthread_t](#) thread)
- int [spthread_suspend_self](#) ()
- int [spthread_continue](#) ([spthread_t](#) thread)
- int [spthread_cancel](#) ([spthread_t](#) thread)
- bool [spthread_self](#) ([spthread_t](#) *thread)
- int [spthread_join](#) ([spthread_t](#) thread, void **retval)
- void [spthread_exit](#) (void *status)

5.7.1 Macro Definition Documentation

5.7.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

5.7.1.2 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 700
```

5.7.1.3 MILLISEC_IN_NANO

```
#define MILLISEC_IN_NANO 100000
```

5.7.1.4 SPTHREAD_RUNNING_STATE

```
#define SPTHREAD_RUNNING_STATE 0
```

5.7.1.5 SPTHREAD_SIG_CONTINUE

```
#define SPTHREAD_SIG_CONTINUE -2
```

5.7.1.6 SPTHREAD_SIG_SUSPEND

```
#define SPTHREAD_SIG_SUSPEND -1
```

5.7.1.7 SPTHREAD_SUSPENDED_STATE

```
#define SPTHREAD_SUSPENDED_STATE 1
```

5.7.1.8 SPTHREAD_TERMINATED_STATE

```
#define SPTHREAD_TERMINATED_STATE 2
```

5.7.2 Typedef Documentation

5.7.2.1 pthread_fn

```
typedef void *(* pthread_fn) (void *)
```

5.7.2.2 spthread_fwd_args

```
typedef struct spthread_fwd_args_st spthread_fwd_args
```

5.7.2.3 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

5.7.2.4 spthread_signal_args

```
typedef struct spthread_signal_args_st spthread_signal_args
```

5.7.3 Function Documentation

5.7.3.1 spthread_cancel()

```
int spthread_cancel (
    spthread_t thread )
```

5.7.3.2 spthread_continue()

```
int spthread_continue (
    spthread_t thread )
```

5.7.3.3 spthread_create()

```
int spthread_create (
    spthread_t * thread,
    const pthread_attr_t * attr,
    pthread_fn start_routine,
    void * arg )
```

5.7.3.4 pthread_exit()

```
void pthread_exit (
    void * status )
```

5.7.3.5 pthread_join()

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

5.7.3.6 pthread_self()

```
bool pthread_self (
    pthread_t * thread )
```

5.7.3.7 pthread_suspend()

```
int pthread_suspend (
    pthread_t thread )
```

5.7.3.8 pthread_suspend_self()

```
int pthread_suspend_self ( )
```

5.8 src/util/spthread.h File Reference

```
#include <pthread.h>
#include <stdbool.h>
```

Data Structures

- struct [pthread_st](#)

Macros

- #define [SIGPTHD](#) SIGUSR1

Typedefs

- typedef struct [pthread_meta_st](#) [pthread_meta_t](#)
- typedef struct [pthread_st](#) [pthread_t](#)

Functions

- int `spthread_create` (`spthread_t` *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg)
- int `spthread_suspend` (`spthread_t` thread)
- int `spthread_suspend_self` ()
- int `spthread_continue` (`spthread_t` thread)
- int `spthread_cancel` (`spthread_t` thread)
- bool `spthread_self` (`spthread_t` *thread)
- int `spthread_join` (`spthread_t` thread, void **retval)
- void `spthread_exit` (void *status)

5.8.1 Macro Definition Documentation

5.8.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

5.8.2 Typedef Documentation

5.8.2.1 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

5.8.2.2 spthread_t

```
typedef struct spthread_st spthread_t
```

5.8.3 Function Documentation

5.8.3.1 spthread_cancel()

```
int spthread_cancel (  
    spthread_t thread )
```

5.8.3.2 spthread_continue()

```
int spthread_continue (  
    spthread_t thread )
```

5.8.3.3 spthread_create()

```
int spthread_create (  
    spthread_t * thread,  
    const pthread_attr_t * attr,  
    void *(*)(void *) start_routine,  
    void * arg )
```

5.8.3.4 pthread_exit()

```
void pthread_exit (
    void * status )
```

5.8.3.5 pthread_join()

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

5.8.3.6 pthread_self()

```
bool pthread_self (
    pthread_t * thread )
```

5.8.3.7 pthread_suspend()

```
int pthread_suspend (
    pthread_t thread )
```

5.8.3.8 pthread_suspend_self()

```
int pthread_suspend_self ( )
```

5.9 pthread.h

[Go to the documentation of this file.](#)

```
00001 #ifndef PTHREAD_H_
00002 #define PTHREAD_H_
00003
00004 #include <pthread.h>
00005 #include <stdbool.h>
00006
00007 // CAUTION: according to `man 7 pthread`:
00008 //
00009 //   On older Linux kernels, SIGUSR1 and SIGUSR2
00010 //   are used. Applications must avoid the use of whichever set of
00011 //   signals is employed by the implementation.
00012 //
00013 // This may not work on other linux versions
00014
00015 // SIGNAL PTHREAD
00016 // NOTE: if within a created pthread you change
00017 // the behaviour of SIGUSR1, then you will not be able
00018 // to suspend and continue a pthread
00019 #define SIGPTHREAD SIGUSR1
00020
00021 // declares a struct, but the internals of the
00022 // struct cannot be seen by functions outside of pthread.c
00023 typedef struct pthread_meta_st pthread_meta_t;
00024
00025 // The pthread wrapper struct.
00026 // Sometimes you may have to access the inner pthread member
00027 // but you shouldn't need to do that
00028 typedef struct pthread_st {
00029     pthread_t thread;
00030     pthread_meta_t* meta;
00031 }
```

```

00031 } spthread_t;
00032
00033 // NOTE:
00034 // None of these are signal safe
00035 // Also note that most of these functions are not safe to suspension,
00036 // meaning that if the thread calling these is an spthread and is suspended
00037 // in the middle of spthread_continue or spthread_suspend, then it may not work.
00038 //
00039 // Make sure that the calling thread cannot be suspended before calling these functions.
00040 // Exceptions to this are spthread_exit(), spthread_self() and if a thread is
00041 // continuing or suspending itself.
00042
00043 // spthread_create:
00044 // this function works similar to pthread_create, except for two differences.
00045 // 1) the created pthread is able to be asynchronously suspended, and continued
00046 //    using the functions:
00047 //        - spthread_suspend
00048 //        - spthread_continue
00049 // 2) The created pthread will be suspended before it executes the specified
00050 //    routine. It must first be continued with `spthread_continue` before
00051 //    it will start executing.
00052 //
00053 // It is worth noting that this function is not signal safe.
00054 // In other words, it should not be called from a signal handler.
00055 //
00056 // to avoid repetition, see pthread_create(3) for details
00057 // on arguments and return values as they are the same here.
00058 int spthread_create(spthread_t* thread,
00059                   const pthread_attr_t* attr,
00060                   void* (*start_routine)(void*),
00061                   void* arg);
00062
00063 // The spthread_suspend function will signal to the
00064 // specified thread to suspend execution.
00065 //
00066 // Calling spthread_suspend on an already suspended
00067 // thread does not do anything.
00068 //
00069 // It is worth noting that this function is not signal safe.
00070 // In other words, it should not be called from a signal handler.
00071 //
00072 // args:
00073 // - pthread_t thread: the thread we want to suspend
00074 //   This thread must be created using the spthread_create() function,
00075 //   if created by some other function, the behaviour is undefined.
00076 //
00077 // returns:
00078 // - 0 on success
00079 // - EAGAIN if the thread could not be signaled
00080 // - ENOSYS if not supported on this system
00081 // - ESRCH if the thread specified is not a valid pthread
00082 int spthread_suspend(spthread_t thread);
00083
00084 // The spthread_suspend_self function will cause the calling
00085 // thread (which should be created by spthread_create) to suspend
00086 // itself.
00087 //
00088 // returns:
00089 // - 0 on success
00090 // - EAGAIN if the thread could not be signaled
00091 // - ENOSYS if not supported on this system
00092 // - ESRCH if the calling thread is not an spthread
00093 int spthread_suspend_self();
00094
00095 // The spthread_continue function will signal to the
00096 // specified thread to resume execution if suspended.
00097 //
00098 // Calling spthread_continue on an already non-suspended
00099 // thread does not do anything.
00100 //
00101 // It is worth noting that this function is not signal safe.
00102 // In other words, it should not be called from a signal handler.
00103 //
00104 // args:
00105 // - spthread_t thread: the thread we want to continue
00106 //   This thread must be created using the spthread_create() function,
00107 //   if created by some other function, the behaviour is undefined.
00108 //
00109 // returns:
00110 // - 0 on success
00111 // - EAGAIN if the thread could not be signaled
00112 // - ENOSYS if not supported on this system
00113 // - ESRCH if the thread specified is not a valid pthread
00114 int spthread_continue(spthread_t thread);
00115
00116 // The spthread_cancel function will send a
00117 // cancellation request to the specified thread.

```



```

00118 //
00119 // as of now, this function is identical to pthread_cancel(3)
00120 // so to avoid repetition, you should look there.
00121 //
00122 // Here are a few things that are worth highlighting:
00123 // - it is worth noting that it is a cancellation __request__
00124 // the thread may not terminate immediately, instead the
00125 // thread is checked whenever it calls a function that is
00126 // marked as a cancellation point. At those points, it will
00127 // start the cancellation procedure
00128 // - to make sure all things are de-allocated properly on
00129 // normal exiting of the thread and when it is cancelled,
00130 // you should mark a deferred de-allocation with
00131 // pthread_cleanup_push(3).
00132 // consider the following example:
00133 //
00134 // void* thread_routine(void* arg) {
00135 //     int* num = malloc(sizeof(int));
00136 //     pthread_cleanup_push(&free, num);
00137 //     return NULL;
00138 // }
00139 //
00140 // this program will allocate an integer on the heap
00141 // and mark that data to be de-allocated on cleanup.
00142 // This means that when the thread returns from the
00143 // routine specified in spthread_create, free will
00144 // be called on num. This will also happen if the thread
00145 // is cancelled and not able to be exited normally.
00146 //
00147 // Another function that should be used in conjunction
00148 // is pthread_cleanup_pop(3). I will leave that
00149 // to you to read more on.
00150 //
00151 // It is worth noting that this function is not signal safe.
00152 // In other words, it should not be called from a signal handler.
00153 //
00154 // args:
00155 // - spthread_t thread: the thread we want to cancel.
00156 // This thread must be created using the spthread_create() function,
00157 // if created by some other function, the behaviour is undefined.
00158 //
00159 // returns:
00160 // - 0 on success
00161 // - ESRCH if the thread specified is not a valid pthread
00162 int spthread_cancel(spthread_t thread);
00163
00164 // Can be called by a thread to get two pieces of information:
00165 // 1. Whether or not the calling thread is an spthread (true or false)
00166 // 2. The spthread_t of the calling thread, if it is an spthread_t
00167 //
00168 // almost always the function will be called like this:
00169 // spthread_t self;
00170 // bool i_am_spthread = spthread_self(&self);
00171 //
00172 // args:
00173 // - spthread_t* thread: the output parameter to get the spthread_t
00174 // representing the calling thread, if it is an spthread
00175 //
00176 // returns:
00177 // - true if the calling thread is an spthread_t
00178 // - false otherwise.
00179 bool spthread_self(spthread_t* thread);
00180
00181 // The equivalent of pthread_join but for spthread
00182 // To make sure all resources are cleaned up appropriately
00183 // s pthreads that are created must at some point have spthread_join
00184 // called on them. Do not use pthread_join on an spthread.
00185 //
00186 // to avoid repetition, see pthread_join(3) for details
00187 // on arguments and return values as they are the same as this function.
00188 int spthread_join(spthread_t thread, void** retval);
00189
00190 // The equivalent of pthread_exit but for spthread
00191 // s pthread_exit must be used by s pthreads instead of pthread_exit.
00192 // Otherwise, calls to s pthread_join or other functions (like s pthread_suspend)
00193 // may not work as intended.
00194 //
00195 // to avoid repetition, see pthread_exit(3) for details
00196 // on arguments and return values as they are the same as this function.
00197 void spthread_exit(void* status);
00198
00199 #endif // SPTHREAD_H_

```

5.10 src/util/sys_call.c File Reference

```
#include "kernel.h"
```

5.11 src/util/sys_call.h File Reference

Functions

- `pid_t s_spawn` (`void *(*func)(void *)`, `char *argv[]`, `int fd0`, `int fd1`)
Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- `pid_t s_waitpid` (`pid_t pid`, `int *wstatus`, `bool nohang`)
Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.
- `int s_kill` (`pid_t pid`, `int signal`)
Send a signal to a particular process.
- `void s_exit` (`void`)
Unconditionally exit the calling process.
- `int s_nice` (`pid_t pid`, `int priority`)
Set the priority of the specified thread.
- `void s_sleep` (`unsigned int ticks`)
Suspends execution of the calling proces for a specified number of clock ticks.

5.11.1 Function Documentation

5.11.1.1 s_exit()

```
void s_exit (
    void )
```

Unconditionally exit the calling process.

5.11.1.2 s_kill()

```
int s_kill (
    pid_t pid,
    int signal )
```

Send a signal to a particular process.

Parameters

<i>pid</i>	Process ID of the target proces.
<i>signal</i>	Signal number to be sent.

Returns

0 on success, -1 on error.

5.11.1.3 s_nice()

```
int s_nice (
    pid_t pid,
    int priority )
```

Set the priority of the specified thread.

Parameters

<i>pid</i>	Process ID of the target thread.
<i>priority</i>	The new priority value of the thread (0, 1, or 2)

Returns

0 on success, -1 on failure.

5.11.1.4 s_sleep()

```
void s_sleep (
    unsigned int ticks )
```

Suspends execution of the calling process for a specified number of clock ticks.

This function is analogous to `sleep(3)` in Linux, with the behavior that the system clock continues to tick even if the call is interrupted. The sleep can be interrupted by a `P_SIGTERM` signal, after which the function will return prematurely.

Parameters

<i>ticks</i>	Duration of the sleep in system clock ticks. Must be greater than 0.
--------------	--

5.11.1.5 s_spawn()

```
pid_t s_spawn (
    void (*)(void *) func,
    char * argv[],
    int fd0,
    int fd1 )
```

Create a child process that executes the function `func`. The child will retain some attributes of the parent.

Parameters

<i>func</i>	Function to be executed by the child process.
-------------	---

Parameters

<i>argv</i>	Null-terminated array of args, including the command name as argv[0].
<i>fd0</i>	Input file descriptor.
<i>fd1</i>	Output file descriptor.

Returns

`pid_t` The process ID of the created child process.

5.11.1.6 s_waitpid()

```
pid_t s_waitpid (
    pid_t pid,
    int * wstatus,
    bool nohang )
```

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

Parameters

<i>pid</i>	Process ID of the child to wait for.
<i>wstatus</i>	Pointer to an integer variable where the status will be stored.
<i>nohang</i>	If true, return immediately if no child has exited.

Returns

`pid_t` The process ID of the child which has changed state on success, -1 on error.

5.12 sys_call.h

[Go to the documentation of this file.](#)

```
00001 /*===== system call functions for interacting with PennOS process creation
00002 =====*/
00012 pid_t s_spawn(void* (*func)(void*), char *argv[], int fd0, int fd1);
00013
00023 pid_t s_waitpid(pid_t pid, int* wstatus, bool nohang);
00024
00032 int s_kill(pid_t pid, int signal);
00033
00037 void s_exit(void);
00038
00039 /*===== system calls for interacting with the scheduler
00040 =====*/
00048 int s_nice(pid_t pid, int priority);
00049
00060 void s_sleep(unsigned int ticks);
```

5.13 test/sched-demo.c File Reference

```
#include <pthread.h>
#include <signal.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdbool.h>
#include "../spthread.h"
```

Macros

- #define `NUM_THREADS` 4
- #define `BUF_SIZE` 4096

Functions

- void `cancel_and_join` (`spthread_t` thread)
- int `main` (void)

5.13.1 Macro Definition Documentation

5.13.1.1 BUF_SIZE

```
#define BUF_SIZE 4096
```

5.13.1.2 NUM_THREADS

```
#define NUM_THREADS 4
```

5.13.2 Function Documentation

5.13.2.1 cancel_and_join()

```
void cancel_and_join (
    spthread_t thread )
```

5.13.2.2 main()

```
int main (
    void )
```


Index

- [_GNU_SOURCE](#)
 - [spthread.c, 13](#)
- [_XOPEN_SOURCE](#)
 - [spthread.c, 13](#)
- [ack](#)
 - [spthread_signal_args_st, 9](#)
- [actual_arg](#)
 - [spthread_fwd_args_st, 7](#)
- [actual_routine](#)
 - [spthread_fwd_args_st, 7](#)
- [BUF_SIZE](#)
 - [sched-demo.c, 23](#)
- [cancel_and_join](#)
 - [sched-demo.c, 23](#)
- [child_meta](#)
 - [spthread_fwd_args_st, 7](#)
- [doc/README.md, 11](#)
- [k_proc_cleanup](#)
 - [kernel.h, 11](#)
- [k_proc_create](#)
 - [kernel.h, 11](#)
- [kernel.h](#)
 - [k_proc_cleanup, 11](#)
 - [k_proc_create, 11](#)
- [main](#)
 - [sched-demo.c, 23](#)
- [meta](#)
 - [spthread_st, 9](#)
- [meta_mutex](#)
 - [spthread_meta_st, 8](#)
- [MILISEC_IN_NANO](#)
 - [spthread.c, 13](#)
- [NUM_THREADS](#)
 - [sched-demo.c, 23](#)
- [pthread_fn](#)
 - [spthread.c, 14](#)
- [README, 1](#)
- [s_exit](#)
 - [sys_call.h, 20](#)
- [s_kill](#)
 - [sys_call.h, 20](#)
- [s_nice](#)
 - [sys_call.h, 21](#)
- [s_sleep](#)
 - [sys_call.h, 21](#)
- [s_spawn](#)
 - [sys_call.h, 21](#)
- [s_waitpid](#)
 - [sys_call.h, 22](#)
- [sched-demo.c](#)
 - [BUF_SIZE, 23](#)
 - [cancel_and_join, 23](#)
 - [main, 23](#)
 - [NUM_THREADS, 23](#)
- [setup_cond](#)
 - [spthread_fwd_args_st, 7](#)
- [setup_done](#)
 - [spthread_fwd_args_st, 7](#)
- [setup_mutex](#)
 - [spthread_fwd_args_st, 8](#)
- [shutup_mutex](#)
 - [spthread_signal_args_st, 9](#)
- [signal](#)
 - [spthread_signal_args_st, 9](#)
- [SIGPTHD](#)
 - [spthread.h, 16](#)
- [spthread.c](#)
 - [_GNU_SOURCE, 13](#)
 - [_XOPEN_SOURCE, 13](#)
 - [MILISEC_IN_NANO, 13](#)
 - [pthread_fn, 14](#)
 - [spthread_cancel, 14](#)
 - [spthread_continue, 14](#)
 - [spthread_create, 14](#)
 - [spthread_exit, 14](#)
 - [spthread_fwd_args, 14](#)
 - [spthread_join, 15](#)
 - [spthread_meta_t, 14](#)
 - [SPTHREAD_RUNNING_STATE, 13](#)
 - [spthread_self, 15](#)
 - [SPTHREAD_SIG_CONTINUE, 13](#)
 - [SPTHREAD_SIG_SUSPEND, 13](#)
 - [spthread_signal_args, 14](#)
 - [spthread_suspend, 15](#)
 - [spthread_suspend_self, 15](#)
 - [SPTHREAD_SUSPENDED_STATE, 13](#)
 - [SPTHREAD_TERMINATED_STATE, 14](#)
- [spthread.h](#)
 - [SIGPTHD, 16](#)
 - [spthread_cancel, 16](#)

- spthread_continue, 16
- spthread_create, 16
- spthread_exit, 16
- spthread_join, 17
- spthread_meta_t, 16
- spthread_self, 17
- spthread_suspend, 17
- spthread_suspend_self, 17
- spthread_t, 16
- spthread_cancel
 - spthread.c, 14
 - spthread.h, 16
- spthread_continue
 - spthread.c, 14
 - spthread.h, 16
- spthread_create
 - spthread.c, 14
 - spthread.h, 16
- spthread_exit
 - spthread.c, 14
 - spthread.h, 16
- spthread_fwd_args
 - spthread.c, 14
- spthread_fwd_args_st, 7
 - actual_arg, 7
 - actual_routine, 7
 - child_meta, 7
 - setup_cond, 7
 - setup_done, 7
 - setup_mutex, 8
- spthread_join
 - spthread.c, 15
 - spthread.h, 17
- spthread_meta_st, 8
 - meta_mutex, 8
 - state, 8
 - suspend_set, 8
- spthread_meta_t
 - spthread.c, 14
 - spthread.h, 16
- SPTHREAD_RUNNING_STATE
 - spthread.c, 13
- spthread_self
 - spthread.c, 15
 - spthread.h, 17
- SPTHREAD_SIG_CONTINUE
 - spthread.c, 13
- SPTHREAD_SIG_SUSPEND
 - spthread.c, 13
- spthread_signal_args
 - spthread.c, 14
- spthread_signal_args_st, 8
 - ack, 9
 - shutup_mutex, 9
 - signal, 9
- spthread_st, 9
 - meta, 9
 - thread, 9
- spthread_suspend
 - spthread.c, 15
 - spthread.h, 17
- spthread_suspend_self
 - spthread.c, 15
 - spthread.h, 17
- SPTHREAD_SUSPENDED_STATE
 - spthread.c, 13
- spthread_t
 - spthread.h, 16
- SPTHREAD_TERMINATED_STATE
 - spthread.c, 14
- src/pennfat.c, 11
- src/pennos.c, 11
- src/util/kernel.c, 11
- src/util/kernel.h, 11, 12
- src/util/spthread.c, 12
- src/util/spthread.h, 15, 17
- src/util/sys_call.c, 20
- src/util/sys_call.h, 20, 22
- state
 - spthread_meta_st, 8
- suspend_set
 - spthread_meta_st, 8
- sys_call.h
 - s_exit, 20
 - s_kill, 20
 - s_nice, 21
 - s_sleep, 21
 - s_spawn, 21
 - s_waitpid, 22
- test/sched-demo.c, 22
- thread
 - spthread_st, 9