PennOS

1.0

Generated by Doxygen 1.10.0

Chapter 1

README

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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.

```

pcb_t* k_proc_create(pcb_t *parent);

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 MILISEC_IN_NANO

```
#define MILISEC_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 spthread_exit()

```
void spthread_exit (
            void * status )
```

5.7.3.5 spthread_join()

```
int spthread_join (
            spthread_t thread,
            void ** retval )
```

5.7.3.6 spthread_self()

```
bool spthread_self (
            spthread_t * thread )
```

5.7.3.7 spthread_suspend()

```
int spthread_suspend (
            spthread_t thread )
```

5.7.3.8 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

5.8 src/util/spthread.h File Reference

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

Data Structures

- struct spthread_st

Macros

- #define SIGPTHD SIGUSR1

Typedefs

- typedef struct spthread_meta_st spthread_meta_t
- typedef struct spthread_st spthread_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 spthread_exit()

```
void spthread_exit (
             void * status )
```

5.8.3.5 spthread_join()

```
int spthread_join (
             spthread_t thread,
             void ** retval )
```

5.8.3.6 spthread_self()

```
bool spthread_self (
             spthread_t * thread )
```

5.8.3.7 spthread_suspend()

```
int spthread_suspend (
             spthread_t thread )
```

5.8.3.8 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

5.9 spthread.h

Go to the documentation of this file.

```
#ifndef SPTHREAD_H_
#define SPTHREAD_H_

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

// CAUTION: according to `man 7 pthread`:
//
//   On older Linux kernels, SIGUSR1 and SIGUSR2
//   are used.  Applications must avoid the use of whichever set of
//   signals is employed by the implementation.
//
// This may not work on other linux versions

// SIGNAL PTHREAD
// NOTE: if within a created spthread you change
// the behaviour of SIGUSR1, then you will not be able
// to suspend and continue a spthread
#define SIGPTHD SIGUSR1

// declares a struct, but the internals of the
// struct cannot be seen by functions outside of spthread.c
typedef struct spthread_meta_st spthread_meta_t;

// The spthread wrapper struct.
// Sometimes you may have to access the inner pthread member
// but you shouldn't need to do that
typedef struct spthread_st {
  pthread_t thread;
  spthread_meta_t* meta;
} spthread_t;

// NOTE:
// None of these are signal safe
// Also note that most of these functions are not safe to suspension,
// meaning that if the thread calling these is an spthread and is suspended
// in the middle of spthread_continue or spthread_suspend, then it may not work.
//
// Make sure that the calling thread cannot be suspended before calling these functions.
// Exceptions to this are spthread_exit(), spthread_self() and if a thread is
// continuing or suspending itself.

// spthread_create:
// this function works similar to pthread_create, except for two differences.
// 1) the created pthread is able to be asychronously suspended, and continued
//    using the functions:
//      - spthread_suspend
//      - spthread_continue
// 2) The created pthread will be suspended before it executes the specified
//    routine. It must first be continued with `spthread_continue` before
//    it will start executing.
//
// It is worth noting that this function is not signal safe.
// In other words, it should not be called from a signal handler.
//
// to avoid repetition, see pthread_create(3) for details
// on arguments and return values as they are the same here.
int spthread_create(spthread_t* thread,
                    const pthread_attr_t* attr,
                    void* (*start_routine)(void*),
                    void* arg);

// The spthread_suspend function will signal to the
// specified thread to suspend execution.
//
// Calling spthread_suspend on an already suspended
// thread does not do anything.
//
// It is worth noting that this function is not signal safe.
// In other words, it should not be called from a signal handler.
//
// args:
// - pthread_t thread: the thread we want to suspend
//   This thread must be created using the spthread_create() function,
//   if created by some other function, the behaviour is undefined.
//
// returns:
// - 0 on success
// - EAGAIN if the thread could not be signaled
// - ENOSYS if not supported on this system
// - ESRCH if the thread specified is not a valid pthread
int spthread_suspend(spthread_t thread);

// The spthread_suspend_self function will cause the calling
// thread (which should be created by spthread_create) to suspend
// itself.
//
// returns:
// - 0 on success
// - EAGAIN if the thread could not be signaled
// - ENOSYS if not supported on this system
// - ESRCH if the calling thread is not an spthread
int spthread_suspend_self();

// The spthread_continue function will signal to the
// specified thread to resume execution if suspended.
//
// Calling spthread_continue on an already non-suspended
// thread does not do anything.
//
// It is worth noting that this function is not signal safe.
// In other words, it should not be called from a signal handler.
//
// args:
// - spthread_t thread: the thread we want to continue
//   This thread must be created using the spthread_create() function,
//   if created by some other function, the behaviour is undefined.
//
// returns:
// - 0 on success
// - EAGAIN if the thread could not be signaled
// - ENOSYS if not supported on this system
// - ESRCH if the thread specified is not a valid pthread
int spthread_continue(spthread_t thread);

// The spthread_cancel function will send a
// cancellation request to the specified thread.
```

```
//
// as of now, this function is identical to pthread_cancel(3)
// so to avoid repitition, you should look there.
//
// Here are a few things that are worth highlighting:
// - it is worth noting that it is a cancellation __request__
//   the thread may not terminate immediately, instead the
//   thread is checked whenever it calls a function that is
//   marked as a cancellation point. At those points, it will
//   start the cancellation procedure
// - to make sure all things are de-allocated properly on
//   normal exiting of the thread and when it is cancelled,
//   you should mark a defered de-allocation with
//   pthread_cleanup_push(3).
//   consider the following example:
//
//     void* thread_routine(void* arg) {
//        int* num = malloc(sizeof(int));
//        pthread_cleanup_push(&free, num);
//        return NULL;
//     }
//
//    this program will allocate an integer on the heap
//    and mark that data to be de-allocated on cleanup.
//    This means that when the thread returns from the
//    routine specified in spthread_create, free will
//    be called on num. This will also happen if the thread
//    is cancelled and not able to be exited normally.
//
//    Another function that should be used in conjunction
//    is pthread_cleanup_pop(3). I will leave that
//    to you to read more on.
//
// It is worth noting that this function is not signal safe.
// In other words, it should not be called from a signal handler.
//
// args:
// - spthread_t thread: the thread we want to cancel.
//   This thread must be created using the spthread_create() function,
//   if created by some other function, the behaviour is undefined.
//
// returns:
// - 0 on success
// - ESRCH if the thread specified is not a valid pthread
int spthread_cancel(spthread_t thread);

// Can be called by a thread to get two peices of information:
// 1. Whether or not the calling thread is an spthread (true or false)
// 2. The spthread_t of the calling thread, if it is an spthread_t
//
// almost always the function will be called like this:
// spthread_t self;
// bool i_am_spthread = spthread_self(&self);
//
// args:
// - spthread_t* thread: the output parameter to get the spthread_t
//   representing the calling thread, if it is an spthread
//
// returns:
// - true if the calling thread is an spthread_t
// - false otherwise.
bool spthread_self(spthread_t* thread);

// The equivalent of pthread_join but for spthread
// To make sure all resources are cleaned up appropriately
// spthreads that are created must at some ppoint have spthread_join
// called on them. Do not use pthread_join on an spthread.
//
// to avoid repetition, see pthread_join(3) for details
// on arguments and return values as they are the same as this function.
int spthread_join(spthread_t thread, void** retval);

// The equivalent of pthread_exit but for spthread
// spthread_exit must be used by spthreads instead of pthread_exit.
// Otherwise, calls to spthread_join or other functions (like spthread_suspend)
// may not work as intended.
//
// to avoid repetition, see pthread_exit(3) for details
// on arguments and return values as they are the same as this function.
void spthread_exit(void* status);

#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

pid	Process ID of the target proces.
signal	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

pid	Process ID of the target thread.
priority	The new priorty 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 proces 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

ticks	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

func	Function to be executed by the child process.

Parameters

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

pid	Process ID of the child to wait for.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	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.

```
/*============================ system call functions for interacting with PennOS process creation
===================================*/
pid_t s_spawn(void* (*func)(void*), char *argv[], int fd0, int fd1);

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

int s_kill(pid_t pid, int signal);

void s_exit(void);

/*============================ system calls for interacting with the scheduler
===================================*/

int s_nice(pid_t pid, int priority);

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