Pthread (9A)

- Pthread

Young Won Lim
02/25/2013

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Young Won Lim
02/25/2013

Pthread Creation and Termination

* pointer type

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

                       void * arg                  );
```

stores the **ID** of the created thread in the location referenced by **thread**

```
void pthread_join(pthread_t thread, void **status);
```

to **wait** for a thread to **terminate**

```
void pthread_detach(pthread_t thread);
```

an alternative to **pthread_join**() to reclaim storage for a thread

```
void pthread_exit(void *status);

int pthread_cancel(pthread_t thread);

int pthread_attr_init(pthread_attr_t *attr);

int pthread_attr_destroy(pthread_attr_t *attr);
```

Creating a Default Thread

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

stores the **ID** of the created thread
in the location referenced by **thread**

```
pthread_t          tid;
pthread_attr_t     tattr;
extern void *      start_routine     (void *arg);
void *             arg;
int                ret;
```

```
ret = pthread_create(&tid, NULL, start_routine, arg);
```

Create default attributes

```
ret = pthread_attr_init(&tattr);
ret = pthread_create(&tid, &tattr, start_routine, arg);
```

Default Attributes

scope	PTHREAD_SCOPE_PROCESS	: unbounded
detachstate	PTHREAD_CREATE_JOINABLE	: nondetatched
stackaddr	NULL	: default stack
stacksize	1 megabyte	: default stack size
inheritsched	PTHREAD_INHERIT_SCHED	: inherit parent's priority

Waiting for a thread to terminate

```
void pthread_join(pthread_t thread, void **status);
```

to **wait** for a thread to **terminate**

works only for target threads **that are nondetached**

blocks the calling thread
until the specified thread **terminates**.

thread

```
pthread_t        tid;
pthread_attr_t   tattr;
extern void *    start_routine    (void *arg);
void *           arg;
int              ret;
```

```
ret = pthread_create(&tid, NULL, start_routine, arg);
```

```
ret = pthread_attr_init(&tattr);
ret = pthread_create(&tid, &tattr, start_routine, arg);
```

```
pthread_join(tid, NULL);
```

```
int * status;
    pthread_join(tid, &status);
```

status

exit code of the defunct thread

Detaching a thread

```
void pthread_detach(pthread_t thread);
```

an alternative to pthread_join() to reclaim storage for a thread that is created with a detachstate attribute set to PTHREAD_CREATE_JOINABLE.

If tid has not terminated, pthread_detach() does not cause it to terminate.

```
pthread_t        tid;
pthread_attr_t   tattr;
extern void *    start_routine    (void *arg);
void *           arg;
int              ret;
```

```
ret = pthread_create(&tid, NULL, start_routine, arg);
```

```
ret = pthread_attr_init(&tattr);
ret = pthread_create(&tid, &tattr, start_routine, arg);
```

```
pthread_join(tid, NULL);
```

```
int * status;
pthread_join(tid, &status);
```

status

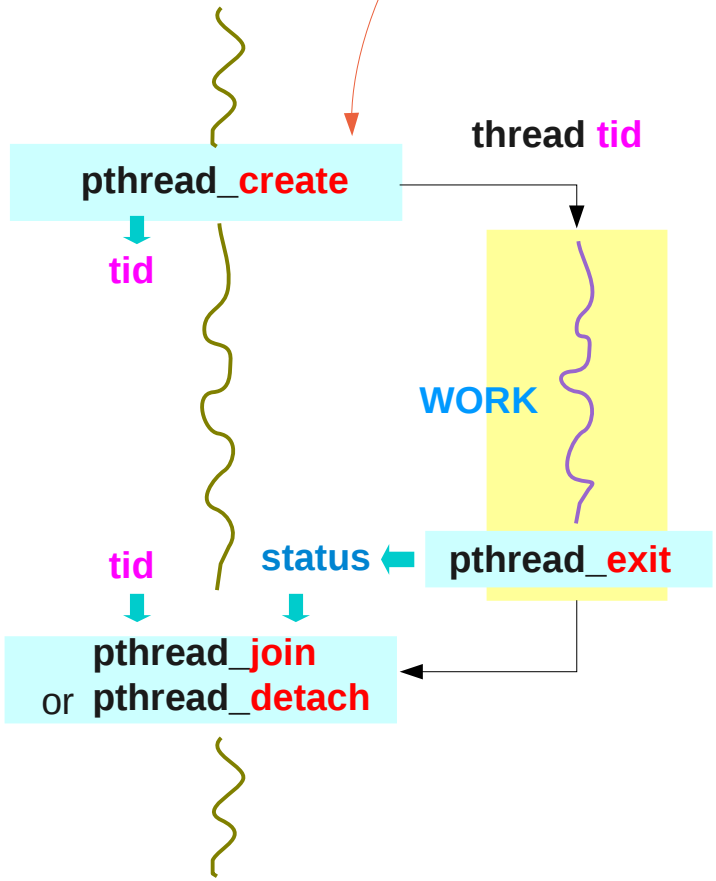
exit code of the defunct thread

```
pthread_detach( tid );
```

Joining and Detaching

Default Attributes		
scope	PTHREAD_SCOPE_PROCESS	: unbounded
detachstate	PTHREAD_CREATE_JOINABLE	: nondetatched
stackaddr	NULL	: default stack
stacksize	1 megabyte	: default stack size
inheritsched	PTHREAD_INHERIT_SCHED	: inherit parent's priority

* If a thread is created as detached, it can never be joined.



a pthread attribute variable of the **pthread_attr_t** type
pthread_attr_init()
pthread_attr_setdetachstate()

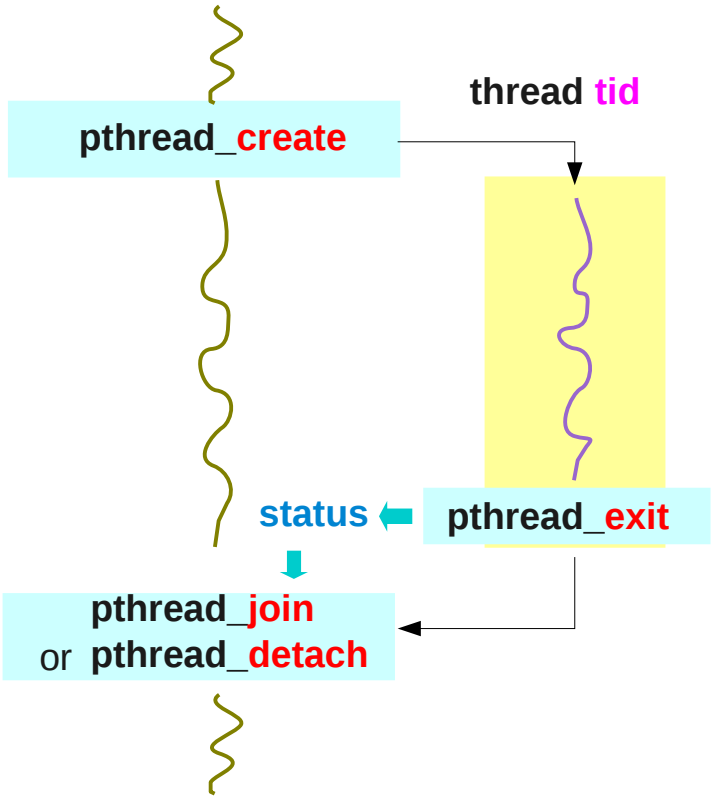
When done,
pthread_attr_destroy()

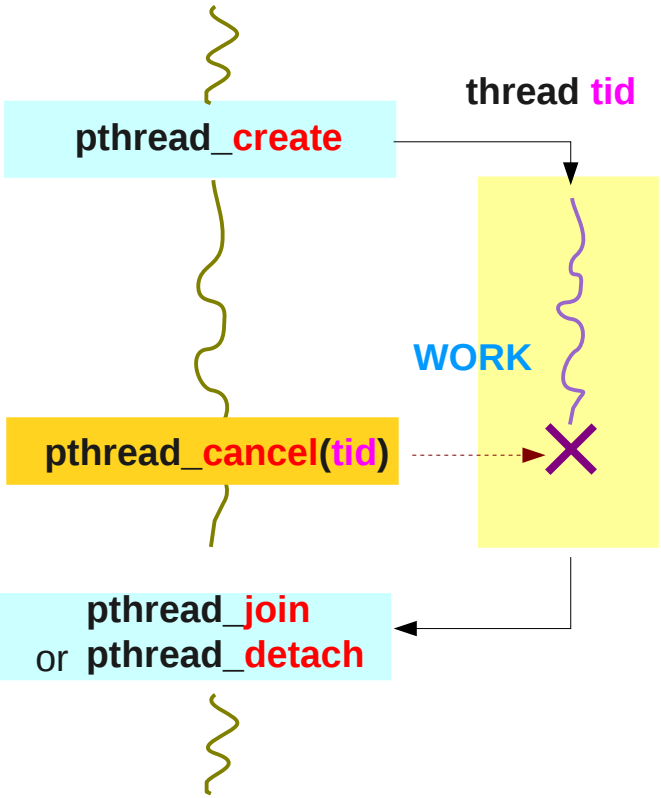
Terminate a thread

void **pthread_exit**(void ***status**);

int **pthread_cancel**(pthread_t **thread**);

A thread is terminated
- By returning from its first (outermost) procedure, the threads start routine;
- By calling **pthread_exit**(), supplying an **exit status**
- By termination with **pthread_cancel**()

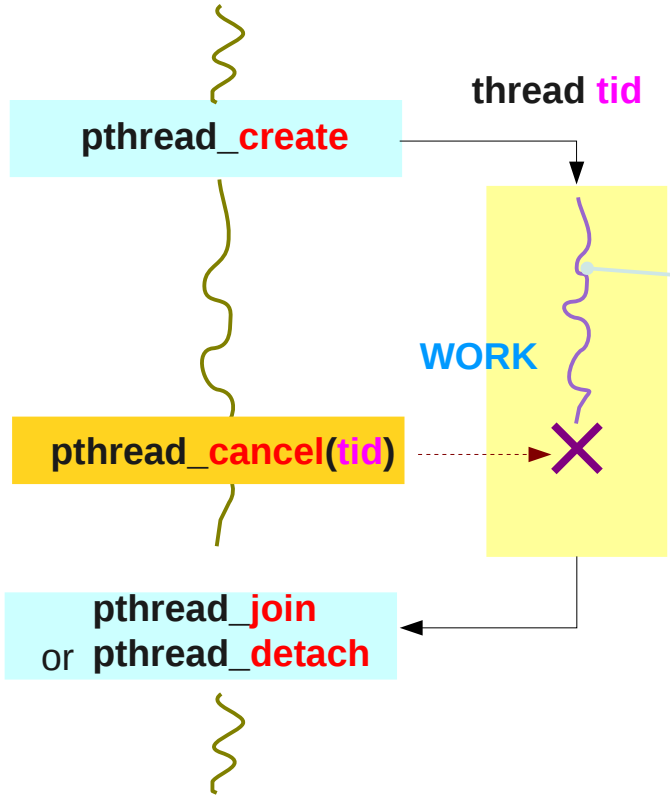





```
int status;
    pthread_exit(&status);
```

```
ret = pthread_cancel(tid);
```

Thread Cancellation



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

States
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DISABLE.

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

Types
PTHREAD_CANCEL_DEFERRED
deferred until a cancellation point
PTHREAD_CANCEL_ASYNCHRONOUS.
cancel immediately even
not at a cancellation point

void **pthread_testcancel** (void);

behaves like a cancellation point function

cancellation point function
read(), write(), open(), close(), fcntl(), sleep(), wait(), waitpid(), pthread_join()

At a cancellation point,
1. always canceled immediately

Not at a cancellation point
2. Deferred – until a cancellation point
3. Asynchronous – immediately canceled

Mutex Variable

```
int pthread_mutex_destroy (pthread_mutex_t *mutex);

int pthread_mutex_init (pthread_mutex_t *restrict mutex,
                        const pthread_mutexattr_t *restrict attr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;


int pthread_mutexattr_destroy (pthread_mutexattr_t *attr);

int pthread_mutexattr_init (pthread_mutexattr_t *attr);


int pthread_mutex_lock (pthread_mutex_t *mutex);

int pthread_mutex_trylock (pthread_mutex_t *mutex);

int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

Creating & Destroying Mutex Variables

Static Initialization

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Dynamic Initialization

```
pthread_mutex_t mutex;
pthread_mutexattr_t attr;
```

```
pthread_mutexattr_settype (&attr , type);
```

PTHREAD_MUTEX_TIMED_NP
PTHREAD_MUTEX_RECURSIVE_NP
PTHREAD_MUTEX_ERRORCHECK_NP

```
pthread_mutex_init (&mutex, &attr);
```

Protocol: Specifies the protocol used to prevent priority inversions for a mutex.
Prioceiling: Specifies the priority ceiling of a mutex.
Process-shared: Specifies the process sharing of a mutex.

```
pthread_mutexattr_destroy (&attr);

pthread_mutex_destroy (&mutex);
```

Locking & Unlocking Mutex Variables

int **pthread_mutex_lock** (pthread_mutex_t ***mutex**);

to **acquire** a lock on the specified mutex variable.
already locked --> block until the mutex is unlocked.

int **pthread_mutex_trylock** (pthread_mutex_t ***mutex**);

to **attempt** to lock a mutex.
already locked --> return immediately with a "**busy**"
Useful in preventing deadlock conditions,
as in a priority-inversion situation.

int **pthread_mutex_unlock** (pthread_mutex_t ***mutex**);

Required after a thread has completed
An **error** will be returned if:
If the mutex was **already unlocked**
If the mutex is **owned by another thread**

Conditional Variable

```
int pthread_cond_destroy (pthread_cond_t *cond);

int pthread_cond_init (pthread_cond_t *restrict cond,
                       const pthread_condattr_t *restrict attr);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;


int pthread_condattr_destroy (pthread_condattr_t *attr);

int pthread_condattr_init (pthread_condattr_t *attr);


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

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

Reference

References

[1] http://en.wikipedia.org/
[2] http://www.tldp.org/LDP/lpg/node46.html