Recap

Shared resources

- Code, global data
- Open files, current working directory
- Credentials

Thread-specific resources

- Thread ID
- Registers, stack
- Priority
- `errno` (error codes)

Thread creation

```
#include <pthread.h>

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

- First argument must contain address of valid, writable location.
- Types (`pthread_t`, `pthread_attr_t`) are *opaque*, i.e., fields are not known / cannot be accessed.
- Compiling: `gcc -pthread ...` OR `gcc ... -lpthread`

Thread termination

- By calling `pthread_exit(void *retval)`
- By returning from `start_routine()`
  - equivalent to implicit call to `pthread_exit()` (for all threads other than the thread in which `main()` was first invoked)
  - return value of `start_routine()` $\equiv$ thread's exit status
- By cancelling (killing) using `pthread_cancel()` ‡
- When any thread calls `exit()` or `exec()`
- When 'main' thread returns from `main()`, without calling `pthread_exit()`

} *all* threads in process terminate

- If 'main' thread calls `pthread_exit()`, process exits with status 0 after termination of last thread

Thread termination

```
void pthread_exit(void *retval);
```

- `retval` : return value

  NOTE: *avoid dangling pointers*

  The value pointed to by `retval` *should not be located on the calling thread's stack, since the contents of that stack are undefined after the thread terminates.*

  — from the man page

- Does not release any application visible process resources, i.e., does not release mutexes, close open files, etc.

Other useful functions

```
int pthread_join(pthread_t tid, void **thread_return);
```

- `tid` : calling thread suspended until thread `tid` terminates
- `thread_return` : if not NULL, return value of `tid` is stored in location pointed to by `thread_return`
- analogous to `wait()`

Other useful functions

```
int pthread_join(pthread_t tid, void **thread_return);
```

- `tid` : calling thread suspended until thread `tid` terminates
- `thread_return` : if not NULL, return value of `tid` is stored in location pointed to by `thread_return`
- analogous to `wait()`

```
pthread_t pthread_self(void);
```

```
// return "TRUE" if equal
int pthread_equal(pthread_t tid1, pthread_t tid2);
```

Misc. information

Linux-specific; obtained from Internet sources, subject to confirmation.

- 1:1 correspondence between each pthread and a kernel thread
  - many-to-many correspondence: Solaris, Windows 7
  - many-to-one correspondence: user-level threads
- Thread ID unique only in context of a single process
- `fork()` duplicates only calling thread
- `exec()` from any thread stops all threads in parent process

Synchronisation

```
pthread_mutex_t initialised_mutex = PTHREAD_MUTEX_INITIALIZER,
                uninitalised_mutex;

int pthread_mutex_init(pthread_mutex_t *uninitialised_mutex,
                       const  pthread_mutexattr_t *mutexattr);

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- `pthread_mutex_trylock`
  - if mutex is unlocked, locks mutex
  - if mutex is locked, returns error code EBUSY (does not block)

Thread safe / reentrant functions

- Some functions use static or global variables to save state information across calls, e.g., `strtok()`
  $\Rightarrow$ non-thread safe / non re-entrant
- Thread safe versions: `strtok_r()`

Useful websites

- *Recommended:* `https://computing.llnl.gov/tutorials/pthreads/`
- `http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html`
- Oracle's multithreaded programming guide
  `https://docs.oracle.com/cd/E18752_01/html/816-5137/toc.html`
- `http://people.cs.pitt.edu/~melhem/courses/xx45p/pthread.pdf`
- `https://randu.org/tutorials/threads/`
- Simple examples:
  `https://linuxprograms.wordpress.com/2007/12/29/threads-programming-in-linux-examples/`