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

```c
#include <pthread.h>

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

- **Thread ID** (≡ unsigned long int)
- **Attributes** (default if NULL)
- **Arguments to start_routine**

- **Compiling:** gcc -pthread ... OR gcc ... -lpthread
Thread termination

- By calling `pthread_exit(void *retval)`
- By returning from `start_routine()`
- By cancelling (killing) using `pthread_cancel()`
- When any thread calls `exit()`, or main thread returns from `main()` (all threads in process terminate)
Thread termination

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

- `retval`: return value
- **NOTE:** *avoid dangling pointers*
Thread attributes

- **Initialisation:**

  ```c
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  ```

- **Getting / setting thread attributes:**

  ```c
  int pthread_attr_setdetachstate(&attr, int detachstate);
  int pthread_attr_setguardsize(&attr, size_t guardsize);
  int pthread_attr_setinheritsched(&attr, int inheritsched);
  int pthread_attr_setschedparam(&attr, const struct sched_param *
      *param);
  int pthread_attr_setschedpolicy(&attr, int policy);
  int pthread_attr_setscope(&attr, int contentionscope);
  int pthread_attr_setstackaddr(&attr, void *stackaddr);
  int pthread_attr_setstacksize(&attr, size_t stacksize);
  ```
Thread attributes

- detached state: `PTHREAD_CREATE_JOINABLE, PTHREAD_CREATE_DETACHED`
- scheduling attributes: `PTHREAD_INHERIT_SCHED, PTHREAD_EXPLICIT_SCHED`
- scheduling policy: `SCHED_FIFO, SCHED_RR, SCHED_OTHER`
- scheduling parameters
Other useful functions

```c
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()`
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);

int pthread_equal(pthread_t tid1, pthread_t tid2);
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
int pthread_mutex_init(pthread_mutex_t *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 with error code EBUSY (does not block)
Some functions use static or global variables to save state information across calls, e.g., `strtok()`
⇒ non-thread safe / non re-entrant

Thread safe versions: `strtok_r()`
1. Matrix multiplication: generate 2 random 1000x1000 matrices $A$ and $B$ and multiply them. Compare the times taken by single-threaded and multi-threaded programs if the matrices are stored in memory / file.
Useful websites

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- https://randu.org/tutorials/threads/

Simple examples: