Threads

- Learning Objectives
  - Define thread
  - Be comfortable using the pthreads APIs to:
    - create threads
    - wait for threads
    - synchronize threads

Process = Address Space + Thread(s) (1)

- A process is composed of two parts:
  - A part that keeps track of "stuff": Address space
  - A dynamic part: Thread
- Address space:
  - A "place" in which execution happens.
  - The set of addresses (e.g., memory locations) to which a running computation has access.
  - An address space can be physical (addresses map directly to locations in the hardware) or virtual (addresses are "make believe" but get translated into locations in hardware).
  - Address spaces provide protection boundaries.

Process = Address Space + Thread(s) (2)

- A process is composed of two parts:
  - A part that keeps track of "stuff": Address space
  - A dynamic part: Thread
- Thread:
  - A logical flow of control
  - Execution state
- A process has one address space and one or more threads in it.
- Threads share the address space, i.e., memory, so you need to synchronize access to memory between threads.

Pthreads

- Pthreads is a standard interface to threads.
  - Specified by POSIX
- Includes APIs for different aspects of threads:
  - Thread routines (e.g., create, exit, join)
  - Attribute object routines (get and set thread attributes)
  - Mutex routines
  - Condition variable routines
  - Read/write lock routines
  - Per-thread context routines – manage per-thread data
  - Cleanup routines

Thread Routines

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

Mutex Routines

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

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Condition Variable Routines

```
int pthread_cond_init(pthread_cond_t *cond,
    const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond,
    pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond,
    pthread_mutex_t *mutex,
    const struct timespec *abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);
```

Wrapping Up

- Threads are a unit of execution within an address space.
- Since threads share state inside the address space, you almost always need synchronization.
- The `pthreads` library provides an implementation of threads plus a set of synchronization primitives.
- Don't stress if you aren't fully comfortable with `pthreads`; we will practice using the library on Thursday.