Pthreads

- Pthreads is a POSIX standard for describing a thread model, it specifies the API and the semantics of the calls.
- Model popular – nowadays practically all major thread libraries on Unix systems are Pthreads-compatible

Preliminaries

- Include `pthread.h` in the main file
- Compile program with `-lpthread`
  - `gcc -o test test.c -lpthread`
  - may not report compilation errors otherwise but calls will fail
- Good idea to check return values on common functions

Thread creation

- Types: `pthread_t` – type of a thread
- Some calls:

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

  - Call `pthread_exit` in main, don't just fall through;
  - most likely you wouldn't need `pthread_join`
  - detached threads are those which cannot be joined (can also set this at creation)

Contention Scope

- *Contention scope* is the POSIX term for describing bound and unbound threads
- A bound thread is said to have *system contention scope*
  - i.e., it contends with all threads in the system
- An unbound thread has *process contention scope*
  - i.e., it contends with threads in the same process
- You can experiment in your project with this, but for most practical reasons use bound threads (system scope)
  - Solaris LWP switching cheap, Linux is one-to-one anyways…

Attributes

- Type: `pthread_attr_t` (see `pthread_create`)
- attributes define the state of the new thread
- attributes: system scope, joinable, stack size, inheritance… you can use default behaviors with `NULL` in `pthread_create()`

```
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
pthread_attr_{set/get}{attribute}
```

- Example:

```
pthread_attr_t attr;
pthread_attr_init(&attr); // Needed!!!
pthread_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_create(NULL, &attr, foo, NULL);
```

Pthread Mutexes

- Type: `pthread_mutex_t`

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

- Attributes: for shared mutexes/condition vars among processes, for priority inheritance, etc.
  - use defaults

Condition variables

- Type `pthread_cond_t`

```
int pthread_cond_init(pthread_cond_t *cond,
                      const pthread_condattr_t *attr);
int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
int pthread_cond_singal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Some rules…

- Shared data should always be accessed through a single mutex (write a comment)
- Think of a boolean condition (expressed in terms of program variables) for each condition variable. Every time the value of the boolean condition may have changed, call Broadcast for the condition variable
  - Only call Signal when you are absolutely certain that *any and only one* waiting thread can enter the critical section
- Globally order locks, acquire in order in all threads

- see examples from web page, pthreads pages/books
- project 1 discussion