Quick Introduction to Pthreads

- An introduction to common Pthreads calls and mechanisms
- Useful connection between the abstract threads language used in Birrell and the actual library you will use for your project
- Far from complete—see web page for links, man pages for details, or buy the suggested Pthreads book (a very worthwhile investment)

Pthreads

- Pthreads: POSIX standard describing a thread model
- Popular: practically all major thread libraries on Unix systems today are Pthreads-compatible
- Will see some Pthreads primitives needed for the project (also practical advice that may save hours of frustration)

Preliminaries

- Include files: pthread.h the main file and probably only one you’ll need. See man pages of individual calls for further information
- Very important: compile your programs with -lpthread
  - e.g.,
    
```bash
    gcc -o test test.c -lpthread
  ```
  - otherwise may get Sun threads library where most calls fail (e.g., no thread can be created) although compilation is successful (i.e., the symbols are defined)
- Good idea to create wrappers around a few common functions to check return values

Thread Creation Interface

- Types: pthread_t
  - thread type
- 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(void *status);
- Remarks:
  - you’ll probably never need pthread_join
  - call pthread_exit in main thread instead of just falling off end of main()
  - detached threads - threads that cannot be joined (this can also be set at creation)
**Attribute objects (see pthread_create)**

Attribute objects define state of new thread (scheduling scope, detach state, stack base address, stack size, and more)

- Types: pthread_attr_t

- Some calls:
  - int pthread_attr_init(
    pthread_attr_t *attr);
  - default values: system scope, joinable, arbitrary other defaults
  - int pthread_attr_destroy(
    pthread_attr_t *attr);
  - other calls for various aspects
    - pthread_attr_{set|get}scope
    - pthread_attr_{set|get}detachstate
    - pthread_attr_{set|get}stacksize
    - pthread_attr_{set|get}inheritsched

- Example:
  pthread_attr_t attr;
  pthread_attr_init(&attr); // Needed!!!
  pthread_attr_setdetachstate(&attr,
    PTHREAD_CREATE_DETACHED);
  pthread_attr_setscope(&attr,
    PTHREAD_SCOPE_SYSTEM);
  pthread_create(NULL, &attr, foo, NULL);

- Can pass NULL instead of attribute object to get default behavior (true for thread attribute objects but also mutex and condition variable attribute objects, which we will see shortly)

- Why use attribute objects for initialization?
  - pros and cons:

**Pthreads Mutexes**

- Types: pthread_mutex_t

- Some calls:
  - 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);
  - Mutex (and condition variables) also have attribute object. Their defaults very reasonable, however, and probably won’t need to change them
  - with mutex attributes you can:
    - make mutex shared between processes
    - enforce priority inheritance
    - similar for condition variables

**Condition Variables**

- Types: pthread_cond_t

- Some calls:
  - 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_signal(
    pthread_cond_t *cond);
  - int pthread_cond_broadcast(
    pthread_cond_t *cond);