POSIX Thread Model

```
#include <pthread.h>
#include <errno.h>
```

- `sterror(errno)` returns a text description of errno
- Creating a thread:

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *at
                   void *(*start)(void *),
                   void *arg);
```

- Thread descriptor is pointed to by thread
- Thread attributes specify: detach state, stack size/address, scheduling policy, scheduler parameters, etc... (use NULL to get default)
- Third parameter is a pointer to a procedure that the thread starts executing at – notice signature
- Fourth parameter is argument list to pass to the procedure

Example

```
#include <pthread.h>
#include <stdio.h>

int num = 0;

void *add_one(int *thread_num) {
   num++;
   printf("thread %d  num = %d\n", *thread_num, num);
}

void main() {
   pthread_t *thread;
   int my_id = 0;
   int your_id = 1;
   pthread_create(thread, NULL, add_one, &your_id);
   add_one(&my_id);
   //   pthread_join(*thread, NULL);
   pthread_exit(NULL);
   }
```

- compile: gcc mythread.cc -o mythread -lpthread
- What is the output of this program?

Sleeping/Implementing timers

- You will have to use threads to implement your buyer program even for sequential server version
  - One thread for individual and one for bulk
- Use `usleep` and it sleeps only one thread (example to follow)
- use usleep; it gives you finer control over time (microseconds instead of seconds)
- Need to `#include <unistd.h>`

usleep() Example

```
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>

void *slowthread(void *i) {
int j=0;
while(++j){ //print Hi once every 5 seconds
usleep(5000000);
printf("Hi %d from slowthread\n",j);} }
void *fastthread(void *i) {
int j=0; //print Hi once every 1 second
while(++j){
usleep(1000000);
printf("Hi %d from fastthread\n",j);} }

main() {
 pthread_t *thread;
 int dummy;
 pthread_create(thread,NULL,slowthread,
                  (void *) &dummy);
 fastthread((void *) &dummy);
 pthread_exit(NULL); }
```

Output

```
Hi 1 from fastthread
Hi 2 from fastthread
Hi 3 from fastthread
Hi 4 from fastthread
Hi 1 from slowthread
Hi 5 from fastthread
Hi 6 from fastthread
Hi 7 from fastthread
Hi 8 from fastthread
Hi 9 from fastthread
Hi 2 from slowthread
Hi 10 from fastthread
Hi 11 from fastthread
...
```

- Each sleeps independently without blocking the other

Mutex

```
//declare and initialize mutex lock (global variable)
pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;

// Before you create the new thread, init mutex
pthread_mutex_init(&fastmutex,  NULL);

//now before accessing a shared variable, lock
pthread_mutex_lock(&fastmutex);
//do your thing, then unlock
pthread_mutex_unlock(&fastmutex);

//destroy it at the end of the program
pthread_mutex_destroy(&fastmutex);
```

Other resources

- `http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html` – thanks to Mark for suggesting this tutorial
- Resources page on the class website