Preparation for PThreads and Synchronization API

To prepare for PThreads programming, you will need to study PThreads and the related synchronization API.

Here are the tasks:

1. Compile and run the provided *pthread_simple.cc* program on turing or hopper.
   The command you will use to compile the program:

   g++ -o yourexecutable yoursourcecodefile -lpthread

   For example:

   ```
   g++ -o thread1 thread_simple.cc -lpthread
   ```

2. Study the Pthread-related sections in the slides and textbook.
3. Read the POSIX mutex lock and semaphore in the slides and textbook.

The following URLs contain some additional tutorials (in fact, thread_simple.cc was from one of the tutorials). You don't need to read the entire tutorial. Instead, focus on the APIs for thread creation, waiting for threads to finish, mutex lock and resource cleanup. Note that for this assignment, you don't need to use the condition variables (you will instead use semaphores.)

- http://www.ibm.com/developerworks/library/l-posix1/
- http://www.ibm.com/developerworks/linux/library/l-posix2/?S_TACT=105AGX03&S_CMP=EDU
- https://computing.llnl.gov/tutorials/pthreads/

INFORMATION ON PTHREADS From the Textbook

The examples are from the required textbook "Operating System Concepts" (10th edition) Section 4.4.1 with minor changes. They are included for the sake of completeness of this assignment.

Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization. This is a specification for thread behavior, not an implementation. Operating-system designers may implement the specification in any way they wish. Numerous systems implement the Pthreads specification; most are UNIX-type systems, including Linux and macOS. Although Windows doesn't support Pthreads natively, some third-party implementations for Windows are available.

The C program shown in Figure 4.11 demonstrates the basic Pthreads API for constructing a multithreaded program that calculates the summation of a non-negative integer in a separate thread. In a Pthreads program, separate threads begin execution in a specified function. In Figure 4.11, this is the runner() function. When this program begins, a single thread of control begins in main(). After some initialization, main() creates a second thread that begins control in the runner() function. Both threads share the global data sum.

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

int sum;  /* this data is shared by the thread(s) */
void *runner(void *param);    /* the thread */

int main(int argc, char *argv[])
{
        pthread_t tid;  /* the thread identifier */
        pthread_attr_t attr;   /* set of thread attributes */
        if (argc != 2)
         {
            fprintf(stderr,"usage: a.out <integer value>\n");
            return -1;
        }
        if (atoi(argv[1]) < 0)
        {
           fprintf(stderr,"%d must be>= 0\n",atoi(argv[1]));
          return -1;
        }

        /* set the default attributes */
        pthread_attr_init(&attr);
        /* create the thread */
        pthread_create(&tid,&attr,runner,argv[1]);
        /* wait for the thread to exit */
        pthread_join(tid,NULL);
        printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
    int i, upper= atoi(param);
    sum = 0;
    for (i = 1; i <= upper; i++)
      sum += i;
    pthread_exit (0) ;

}
```

Figure 4.11 Multithreaded C Program using Pthreads API

Let's look more closely at this program. All Pthreads programs must include the pthread.h header file. The statement pthread_t tid declares the identifier for the thread we will create. Each thread has a set of attributes, including stack size and scheduling information. The pthread_attr_t attr declaration represents the attributes for the thread. We set the attributes in the function call pthread_attr_init(&attr). Because we did not explicitly set any attributes, we use the default attributes provided. A separate thread is created with the pthread_create() function call. In addition to passing the thread identifier and the attributes for the thread, we also pass the name of the function where the new thread will begin execution—in this case, the runner() function. Last, we pass the integer parameter that was provided on the command line, argv[1].

At this point, the program has two threads: the initial (or parent) thread in main() and the summation (or child) thread performing the summation operation in the runner() function. This program follows the thread create/join strategy, whereby after creating the summation thread, the parent thread will wait for it to terminate by calling the pthread_join() function. The summation thread will terminate when it calls the function pthread_exit(). Once the summation thread has returned, the parent thread will output the value of the shared data sum.