CS220 - Computer System II
Lab 12

Due: 11/30/2017, 11:59pm

References

- http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html

1 Introduction

In this lab, you will experiment with C threads. Threads are a way for a program to divide itself into two or more simultaneously running tasks. Different threads in the same process share same resources. A process without any threads can be thought of as a process with just one thread (called main thread). Multiple threads in a process (including the main thread) share the same copy of the heap and the code. However, each thread gets its own stack. Therefore, they get their own copies of local variables.

2 Creating and using a thread

Threads are implemented in C using a library called pthread. Each thread's code starts from what is called a worker function. A worker function is a function with return type void * that accepts a single argument with type void *. The void pointer allows the function to accept any argument type or even multiple arguments as a struct.

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

/* Each thread could get a different worker function or the same worker
   function could be used by multiple functions. */
void *worker_function(void *payload) {
  /* Thread's code goes here */
  printf("In thread %d\n", (int) payload);
}

#define NUM_THREADS 3

int main() {
  pthread_t threads [NUM_THREADS];
  int i;

  /* Declare payload here */

  for(i = 0; i < NUM_THREADS; i++) {
    pthread_create(&threads[i], NULL, worker_function, (void *) payload);
  }

  printf("In main thread\n");

  for(i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
  }

  printf("Exiting main thread\n");
  pthread_exit(NULL);
  return(0);
}
```

Compile and run the program. The pthread library must be linked. Use option `-pthread`.

1. Examine the man pages for `pthread_create`, `pthread_join` and `pthread_exit`. Understand the different arguments they accept.
2. What is the order of print statements in the above code?
3. Create a global variable `global_var` and update the variable in the worker function. In each thread, update the variable and print it. If one thread modifies the variable does the change reflect in another thread?
4. Allocate some memory using malloc in the main thread and pass the pointer to the worker thread as the argument. If the memory is modified in main thread, does it reflect in the worker thread? Test and verify.

3 Synchronization between threads

From the previous exercise you will have realized that in order to effectively utilize threads, it is important to synchronize or 'control' each thread's access to shared data when the data is modified. We use what is called mutex to synchronize access between threads. Start with the below code.

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

```
#include <pthread.h>

/* Create a mutex this ready to be locked! */
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

int sum = 0;

void *count(void *param) {
  int i;

  /* Same thread that locks the mutex must unlock it */

  /* pthread_mutex_lock(&m); */

  // Other threads that call lock will have to wait until we call unlock
  for (i = 0; i < 10000000; i++) {
    sum += 1;
  }

  /* pthread_mutex_unlock(&m); */
  return NULL;
}

int main() {
  pthread_t tid1, tid2;
  pthread_create(&tid1, NULL, count, NULL);
  pthread_create(&tid2, NULL, count, NULL);

  //Wait for both threads to finish:
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);

  printf("ARRRRG sum is %d\n", sum);
  return 0;
}
```

1. Compile the above code. Execute it multiple times consecutively (you can press the up arrow to repeat the command). What is your finding?

2. Now, uncomment `mutex_lock` and `mutex_unlock` in the count function. Repeat the previous step. What is your finding? The lock prevents the second thread from

making changes while the first thread is making changes.

4 Final Task

Finally, write a multi-threaded program where one thread accepts an input from the keyboard and the another thread prints out the input. The input read in the first thread must be stored into a global char array so that the second thread can access it. You may reuse code from previous tasks in this lab. Remember to create a Makefile to generate this executable, and make the executable's build occur with the `all` target.

5 Submission

Collect the source files a Makefile to build the executable for the final task.