CPRE 308 Lab 03 Report

Reid Schneyer, Section 09

View on gist.github.com

Summary

In this lab, I learned how to use some of the `pthread` functions from `<pthread.h>`, such as `pthread_create()`, `pthread_join()`, `pthread_cond_*`, and how to use mutex to program concurrently

3.1

1. Adding `sleep(5)` before both of the print statements in `thread1()` and `thread2()` causes them to not get printed. This is because the code exits before the either `sleep()` function gets to exit.
2. The messages in the thread functions do get printed, although the print statement in `main()` does not get executed.
3.

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

void * thread1(void * ptr);
void * thread2(void * ptr);

int main(int argc, char **argv){
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);

    pthread_join(&t1, NULL);
    pthread_join(&t2, NULL);

    printf("Hello from the main thread\n");
}

void* thread1(void *ptr){
    sleep(5);
    printf("Hello from thread1\n");
}
void* thread2(void *ptr){
    sleep(5);
    printf("Hello from thread2\n");
}
```

3.2.1

1. `v=0`
2. `v=-990`. This is because the threads don't get locked/unlocked, and both `increment()` and `decrement()` change the `v` variable whenever possible, instead of one waiting for the other to finish.

3.2.2

```
/* t2.c
   synchronize threads through mutex and conditional variable
   To compile use: gcc -o t2 t2.c -lpthread
*/

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

void* hello();    // define two routines called by threads
void* world();
void* again();  //define new routine called by thread
```

```
/* global variable shared by threads */
pthread_mutex_t mutex;          // mutex
pthread_cond_t done_hello, done_world;  // conditional variables (added done_world)
int h_done = 0;         // testing variables
int w_done = 0;         // added w_done, updated done to h_done

int main (){
    pthread_t tid_hello, tid_world, tid_again; // thread ids

    /*  initialize mutex and cond variable  */
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&done_hello, NULL);
    pthread_cond_init(&done_world, NULL); // new cond var init

    pthread_create(&tid_hello, NULL, hello, NULL); //thread creation
    pthread_create(&tid_world, NULL, world, NULL); //thread creation
    pthread_create(&tid_again, NULL, again, NULL); // new thread creation

    /* main waits for the two threads to finish */
    pthread_join(tid_hello, NULL);
    pthread_join(tid_world, NULL);
    pthread_join(tid_again, NULL); //added

    printf("\n");
    return 0;
}

void* hello() {
    pthread_mutex_lock(&mutex);
    printf("hello ");
    fflush(stdout);     // flush buffer to allow instant print out
    h_done = 1; //updated var name
    pthread_cond_signal(&done_hello);   // signal world() thread
    pthread_mutex_unlock(&mutex);   // unlocks mutex to allow world to print
}


void* world() {
    pthread_mutex_lock(&mutex);

    /* world thread waits until done == 1. */
    while(h_done == 0){ //added curly brackets for personal reasons
        pthread_cond_wait(&done_hello, &mutex);
    }
    printf("world ");
    fflush(stdout);
    w_done = 1; //set second done flag
    pthread_cond_signal(&done_world); //signal to new signal
    pthread_mutex_unlock(&mutex); // unlocks mutex
}

// pretty much the same as the original world() function,
// but with different variables for pthread stuff (and different print)
void* again(){
    pthread_mutex_lock(&mutex);
    while (w_done == 0){
        pthread_cond_wait(&done_world, &mutex);
    }
    printf("again ");
    fflush(stdout);
    pthread_mutex_unlock(&mutex);
}
```

3.3

```
void *producer(void *arg)
{
  int producer_done = 0;
  // not really sure if I should lock/unlock in or out of the while loop
  // can't really tell if it makes a difference
  pthread_mutex_lock(&mut);
  while (!producer_done)
  {
    // pthread_mutex_lock(&mut);
    /* TODO: fill in the code here */
    while (supply > 0){ // while there's still a supply, wait
      pthread_cond_wait(&producer_cv, &mut);
    }

    printf("producer thread produces %d items\n", NUM_ITEMS_PER_PRODUCE);
    fflush(stdout);
    supply = 10; // item "production" occurs here
    pthread_cond_broadcast(&consumer_cv); // tell consumers that more items have been produced
    if (num_cons_remaining < 1){//we're out of consumers, exit the program
      producer_done = 1;
    }
    // pthread_mutex_unlock(&mut);
  }
  printf("producer exiting while loop\n");
  pthread_mutex_unlock(&mut);
  return NULL;
}
```