Name: Alisala Mwamba
Section: D
University ID: 510262798

Lab 3 Report

Summary:
10pts

In this lab, I performed experiments in concurrent programming with the pthreads library. I started with a simple program to ensure two threads' functions finished together. Then, I performed experiments to learn how to synchronize threads that share a common data source using mutexes and conditional variables as synchronization mechanisms for the pthreads. Finally, I worked on a producer-consumer thread problem to run a group of consumers and a single producer in synchronization.

Lab Questions:

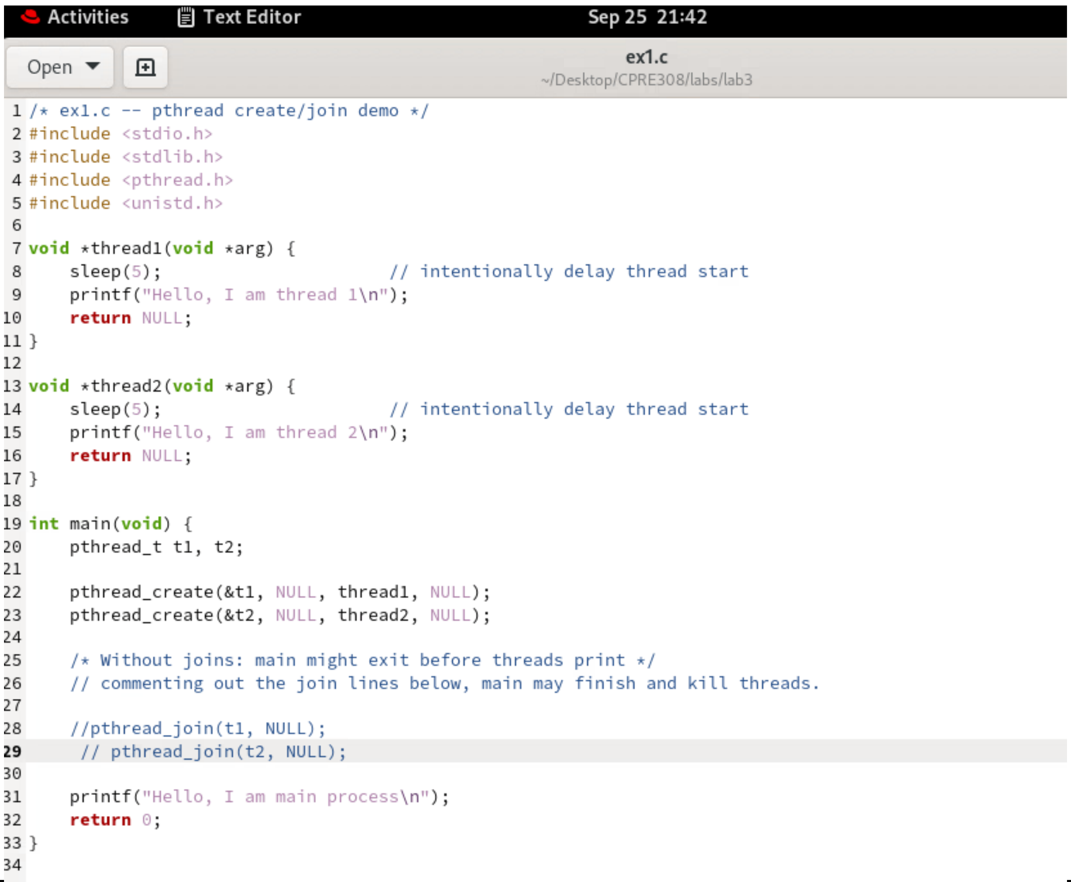
3.1:
6pts To make sure the main terminates before the threads finish,
add a sleep (5) statement in the beginning of the thread functions.
Can you see the threads' output? Why?

I cannot see the thread's output because the sleep () function with pathread_join lines commented out makes the sure the main terminates before the threads finish as in it kills the threads. The output is only "Hello, I am main process" Therefore, the threads output is never printed out because the main is terminated before they are printed. So it order to prevent from happening, we need a way for the main to wait for the threads to finish before it terminates to fix the issue.

2pts Add the two *pthread_join* statements just before the printf statement in main. Pass a value of NULL for the second argument. Recompile and rerun the program. What is the output? Why?

Output:
"Hello, I am thread 1
Hello, I am thread 2
Hello, I am main process" The threads' output is now printed out because the main waits for the two threads to finish.

2pts Include your commented code.



```c
/* ex1.c -- pthread create/join demo */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void *thread1(void *arg) {
    sleep(5);                   // intentionally delay thread start
    printf("Hello, I am thread 1\n");
    return NULL;
}

void *thread2(void *arg) {
    sleep(5);                   // intentionally delay thread start
    printf("Hello, I am thread 2\n");
    return NULL;
}

int main(void) {
    pthread_t t1, t2;

    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);

    /* Without joins: main might exit before threads print */
    // commenting out the join lines below, main may finish and kill threads.

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

    printf("Hello, I am main process\n");
    return 0;
}
```

3.2:

3.2.1:

2pts Compile and run t1.c, what is the output value of v?

V = 0

8pts Delete the *pthread_mutex_lock* and *pthread_mutex_unlock* statement in both increment and decrement threads. Recompile and rerun t1.c, what is the output value of v? Explain why the output is the same, or different.

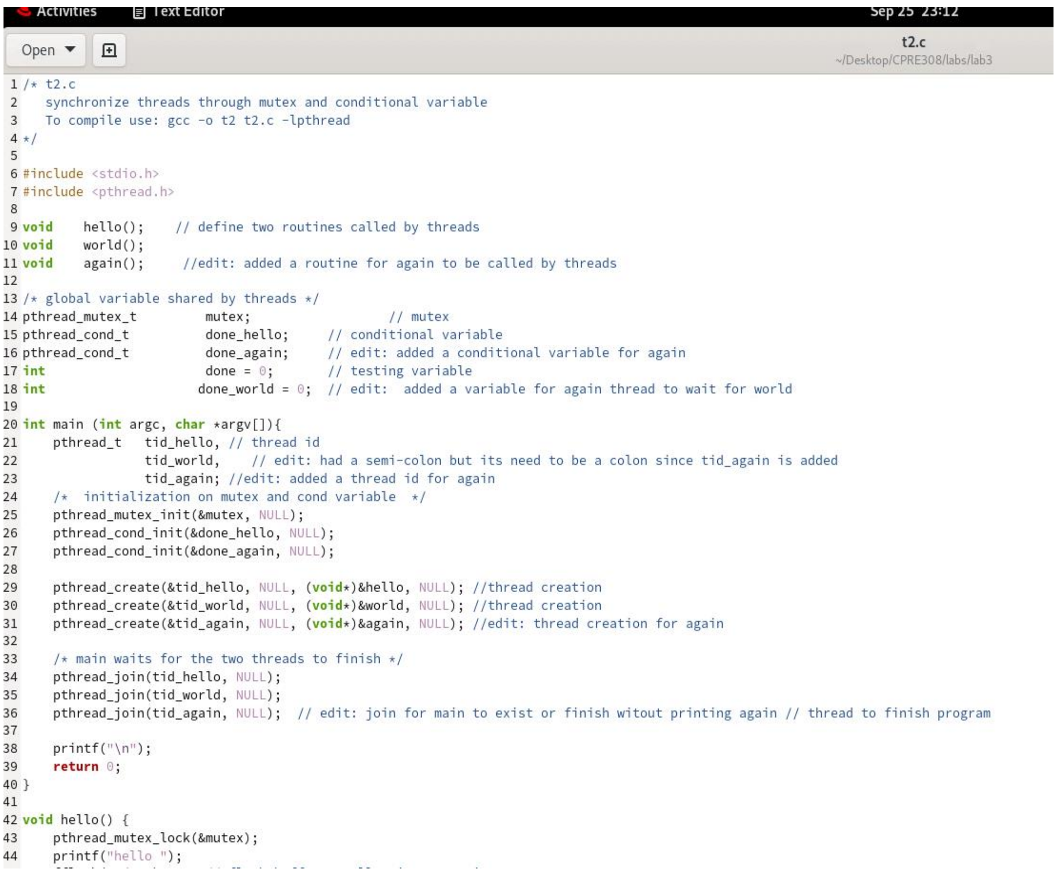
The output value of v is -990. With the pthread_mutex_lock and pthread_mutex_unlock functions, both threads access the global variable v inside a critical section protected by a mutex. The threads run loops 99 times where one thread increments v by 10 every time after which the other thread decrements the same v by 10 for 99 times. Thus, giving us a value of 0. However, when we remove the pthread_mutex_lock and pthread_mutex_unlock functions, both threads are modifying the global variable v concurrently without any thread synchronization, which leads to a race condition. Without locking, the threads can be interrupted between reading and writing the value of v. This causes the threads to overwrite each other's updates. In my case, the decrement function seemed to have dominated the final value since the value of v is more on the negative side. It also means that many of the increment operations were overwritten by the decrement operations.

3.2.2:

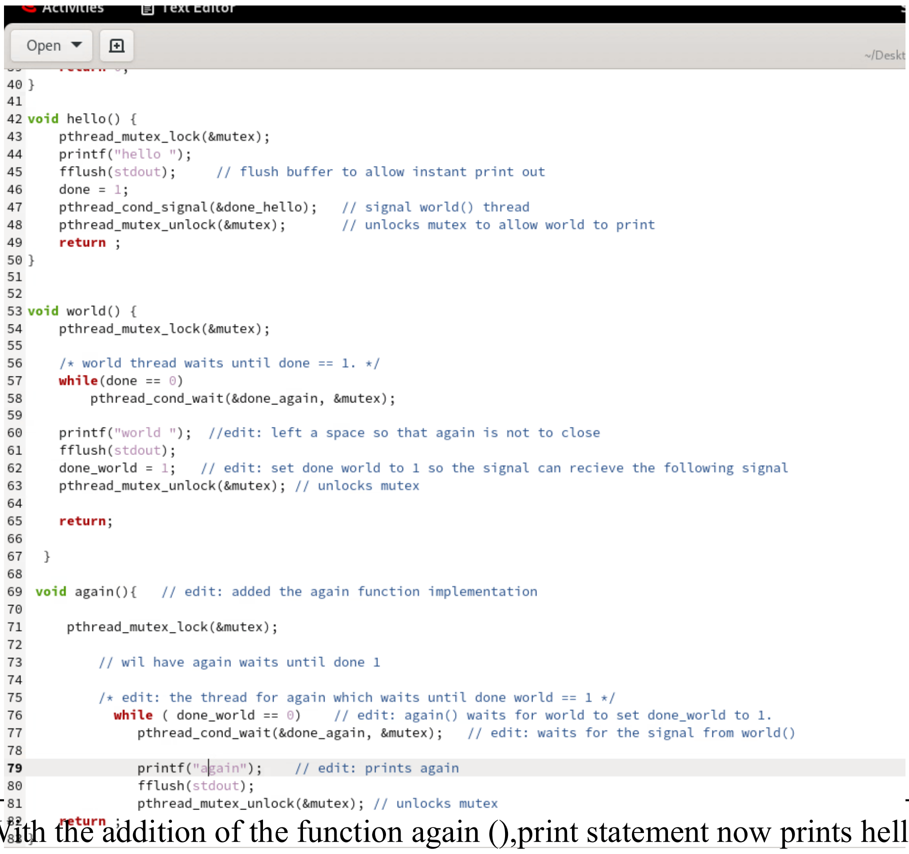
10pts Include your modified code with your lab submission and comment on what you added or changed.

T2.c

Hello sets done = 1 and signals, world waits on hello.



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

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

void    hello();    // define two routines called by threads
void    world();
void    again();     //edit: added a routine for again to be called by threads

/* global variable shared by threads */
pthread_mutex_t         mutex;                  // mutex
pthread_cond_t          done_hello;     // conditional variable
pthread_cond_t          done_again;     // edit: added a conditional variable for again
int                     done = 0;       // testing variable
int                    done_world = 0;  // edit:  added a variable for again thread to wait for world

int main (int argc, char *argv[]){
    pthread_t   tid_hello, // thread id
                tid_world,    // edit: had a semi-colon but its need to be a colon since tid_again is added
                tid_again; //edit: added a thread id for again
    /*  initialization on mutex and cond variable  */
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&done_hello, NULL);
    pthread_cond_init(&done_again, NULL);

    pthread_create(&tid_hello, NULL, (void*)&hello, NULL); //thread creation
    pthread_create(&tid_world, NULL, (void*)&world, NULL); //thread creation
    pthread_create(&tid_again, NULL, (void*)&again, NULL); //edit: thread creation for again

    /* main waits for the two threads to finish */
    pthread_join(tid_hello, NULL);
    pthread_join(tid_world, NULL);
    pthread_join(tid_again, NULL);  // edit: join for main to exist or finish witout printing again // thread to finish program

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

void hello() {
    pthread_mutex_lock(&mutex);
    printf("hello ");
```

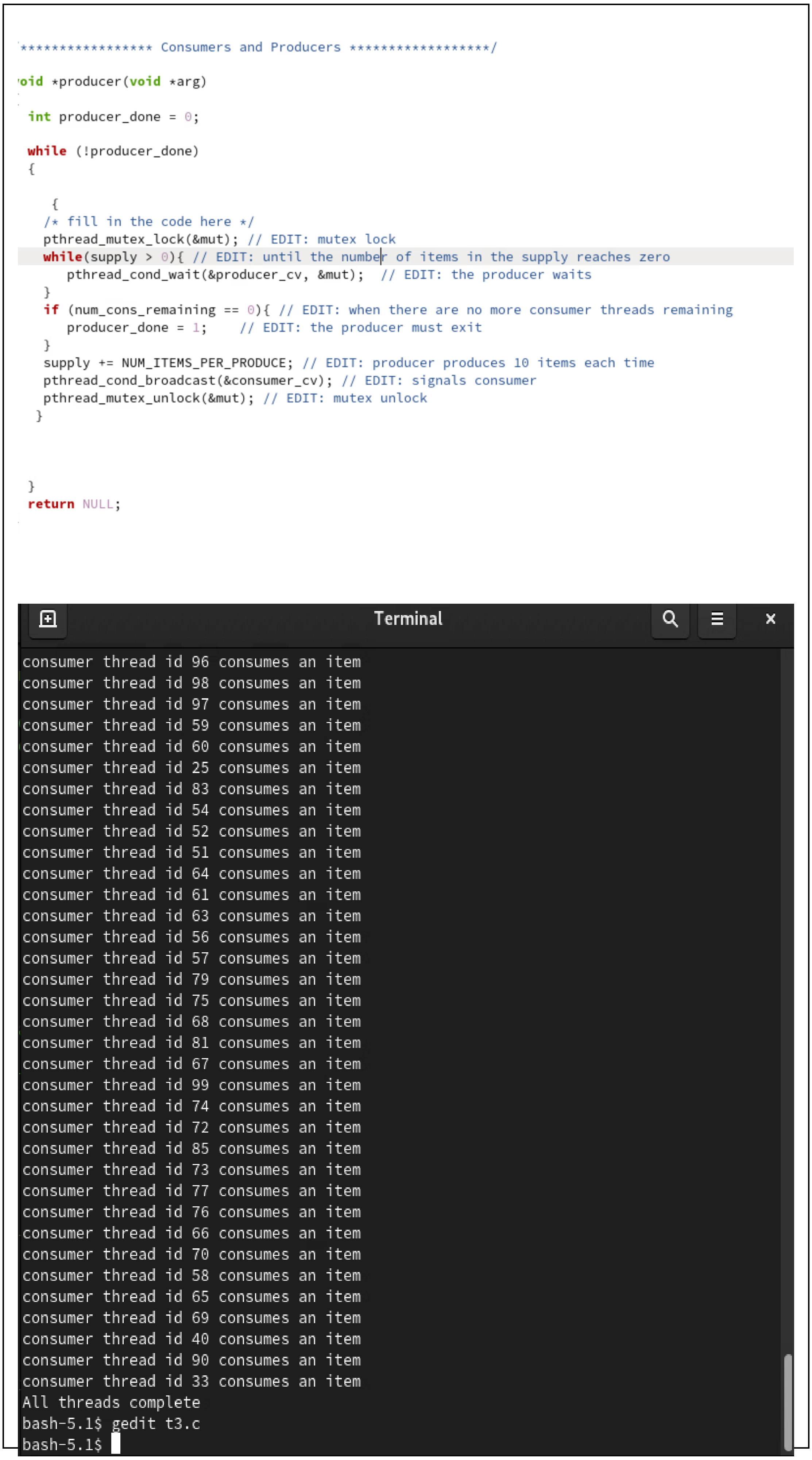


```
}

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


void world() {
    pthread_mutex_lock(&mutex);

    /* world thread waits until done == 1. */
    while(done == 0)
        pthread_cond_wait(&done_again, &mutex);

    printf("world ");  //edit: left a space so that again is not to close
    fflush(stdout);
    done_world = 1;   // edit: set done world to 1 so the signal can recieve the following signal
    pthread_mutex_unlock(&mutex); // unlocks mutex

    return;

  }

 void again(){   // edit: added the again function implementation

     pthread_mutex_lock(&mutex);

         // wil have again waits until done 1

         /* edit: the thread for again which waits until done world == 1 */
           while ( done_world == 0)    // edit: again() waits for world to set done_world to 1.
              pthread_cond_wait(&done_again, &mutex);   // edit: waits for the signal from world()

              printf("again");    // edit: prints again
              fflush(stdout);
              pthread_mutex_unlock(&mutex); // unlocks mutex
    return ;
}
```

With the addition of the function again (),print statement now prints hello world again

3.3:

20pts Include your modified code with your lab submission and comment on what you added or changed.



```c
'***************** Consumers and Producers ******************/

'oid *producer(void *arg)

 int producer_done = 0;

 while (!producer_done)
 {

    {
   /* fill in the code here */
   pthread_mutex_lock(&mut); // EDIT: mutex lock
   while(supply > 0){ // EDIT: until the number of items in the supply reaches zero
      pthread_cond_wait(&producer_cv, &mut);  // EDIT: the producer waits
   }
   if (num_cons_remaining == 0){ // EDIT: when there are no more consumer threads remaining
      producer_done = 1;    // EDIT: the producer must exit
   }
   supply += NUM_ITEMS_PER_PRODUCE; // EDIT: producer produces 10 items each time
   pthread_cond_broadcast(&consumer_cv); // EDIT: signals consumer
   pthread_mutex_unlock(&mut); // EDIT: mutex unlock
  }



 }
 return NULL;
```

Terminal

```
consumer thread id 96 consumes an item
consumer thread id 98 consumes an item
consumer thread id 97 consumes an item
consumer thread id 59 consumes an item
consumer thread id 60 consumes an item
consumer thread id 25 consumes an item
consumer thread id 83 consumes an item
consumer thread id 54 consumes an item
consumer thread id 52 consumes an item
consumer thread id 51 consumes an item
consumer thread id 64 consumes an item
consumer thread id 61 consumes an item
consumer thread id 63 consumes an item
consumer thread id 56 consumes an item
consumer thread id 57 consumes an item
consumer thread id 79 consumes an item
consumer thread id 75 consumes an item
consumer thread id 68 consumes an item
consumer thread id 81 consumes an item
consumer thread id 67 consumes an item
consumer thread id 99 consumes an item
consumer thread id 74 consumes an item
consumer thread id 72 consumes an item
consumer thread id 85 consumes an item
consumer thread id 73 consumes an item
consumer thread id 77 consumes an item
consumer thread id 76 consumes an item
consumer thread id 66 consumes an item
consumer thread id 70 consumes an item
consumer thread id 58 consumes an item
consumer thread id 65 consumes an item
consumer thread id 69 consumes an item
consumer thread id 40 consumes an item
consumer thread id 90 consumes an item
consumer thread id 33 consumes an item
All threads complete
bash-5.1$ gedit t3.c
bash-5.1$ 
```