Name:

Section:

University ID:

Lab 3 Report

Summary:

10pts

In this lab we learned about pthreads and how they work. We created different pthreads and saw how they call a given function by pointer. We used p_thread_cond to implement waits in our code to make a certain process wait for a condition to be met. We also learned about Mutex and how we can use it to limit data access within threads so we don't have multiple threads accessing the same chunk of data. Overall, this lab was a great lab for learning about pthread and how it works.

Lab Questions:

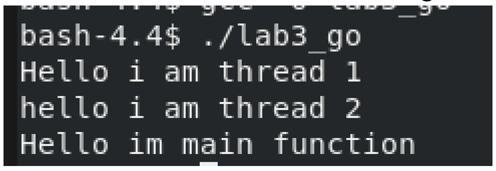
3.1:

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?

No it still does not work. This is because the main thread gets done executing while the threads are still in sleep so the main the other threads get closed once the main function closes.

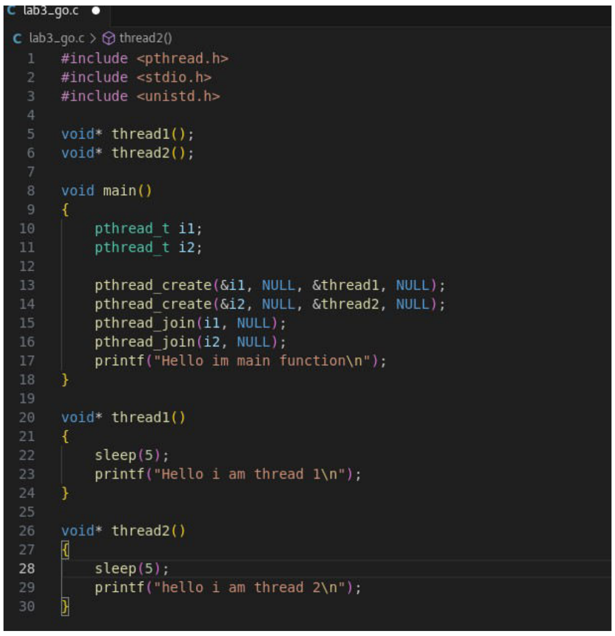
2pts Add the two *pthread_join* statements just before the printf statement in main. Pass a value of NULL for the second argument. Recompile and rerun the program. What is the output? Why?

We get all of the output of the threads. This is because the join function makes the current process wait until the execution of the targeted thread is done executing.



```
bash-4.4$ ./lab3_go
Hello i am thread 1
hello i am thread 2
Hello im main function
```

2pts Include your commented code.



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

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

void main()
{
    pthread_t i1;
    pthread_t i2;

    pthread_create(&i1, NULL, &thread1, NULL);
    pthread_create(&i2, NULL, &thread2, NULL);
    pthread_join(i1, NULL);
    pthread_join(i2, NULL);
    printf("Hello im main function\n");
}

void* thread1()
{
    sleep(5);
    printf("Hello i am thread 1\n");
}

void* thread2()
{
    sleep(5);
    printf("hello i am thread 2\n");
}
```

3.2:

3.2.1:

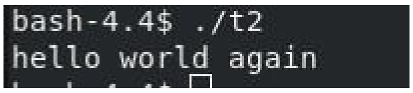
2pts Compile and run t1.c, what is the output value of v?

v=0

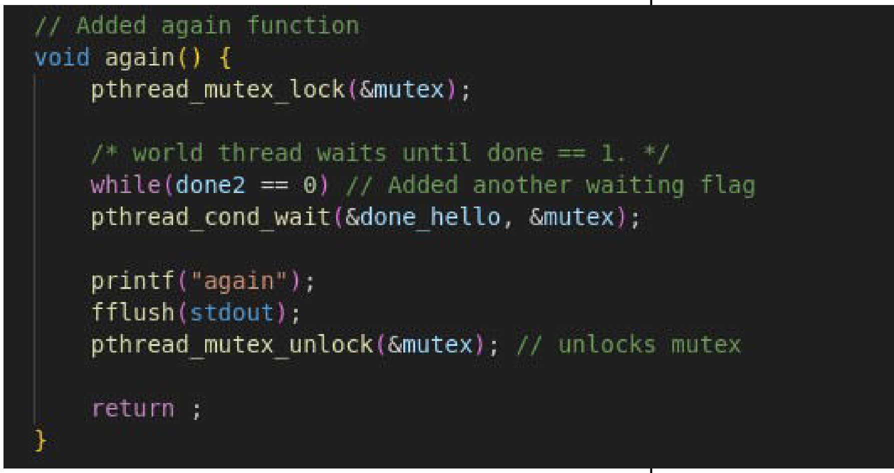
8pts Delete the *pthread_mutex_lock* and *pthread_mutex_unlock* statement in both increment and decrement threads. Recompile and rerun t1.c, what is the output value of v? Explain why the output is the same, or different.

v=-990
It is different because the pthread_mutex_lock makes it so that only one thread can have access to the the proceeding code. Any other threads have to wait until it becomes unlocked to access it. When we remove the lock and unlock functions the two threads are in a race condition where they are both adding and subtracting to v whenever they can.

3.2.2:
10pts Include your modified code with your lab submission and comment on what you added or changed.



```
bash-4.4$ ./t2
hello world again
```

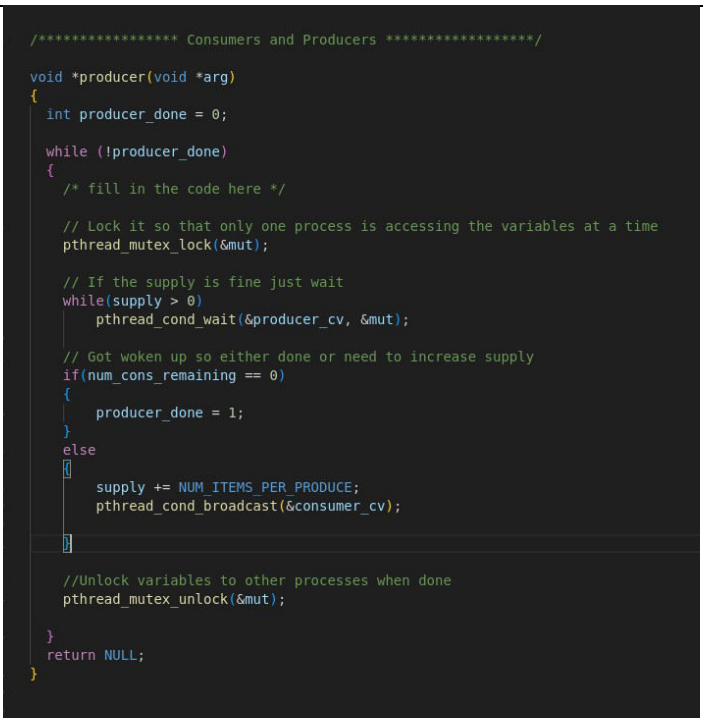


```
// Added again function
void again() {
    pthread_mutex_lock(&mutex);

    /* world thread waits until done == 1. */
    while(done2 == 0) // Added another waiting flag
    pthread_cond_wait(&done_hello, &mutex);

    printf("again");
    fflush(stdout);
    pthread_mutex_unlock(&mutex); // unlocks mutex

    return ;
}
```

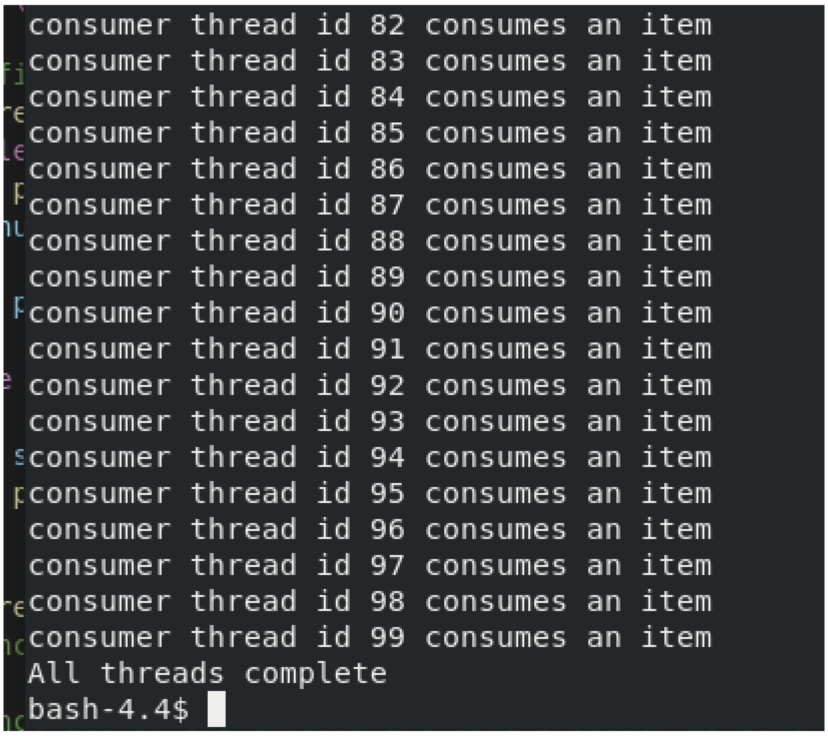
```
// Added again function
void again() {
    pthread_mutex_lock(&mutex);

    /* world thread waits until done == 1. */
    while(done2 == 0) // Added another waiting flag
    pthread_cond_wait(&done_hello, &mutex);

    printf("again");
    fflush(stdout);
    pthread_mutex_unlock(&mutex); // unlocks mutex

    return ;
}
```

3.3:

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



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

void *producer(void *arg)
{
  int producer_done = 0;

  while (!producer_done)
  {
    /* fill in the code here */

    // Lock it so that only one process is accessing the variables at a time
    pthread_mutex_lock(&mut);

    // If the supply is fine just wait
    while(supply > 0)
        pthread_cond_wait(&producer_cv, &mut);

    // Got woken up so either done or need to increase supply
    if(num_cons_remaining == 0)
    {
        producer_done = 1;
    }
    else
    {
        supply += NUM_ITEMS_PER_PRODUCE;
        pthread_cond_broadcast(&consumer_cv);

    }

    //Unlock variables to other processes when done
    pthread_mutex_unlock(&mut);

  }
  return NULL;
}
```



```
consumer thread id 82 consumes an item
consumer thread id 83 consumes an item
consumer thread id 84 consumes an item
consumer thread id 85 consumes an item
consumer thread id 86 consumes an item
consumer thread id 87 consumes an item
consumer thread id 88 consumes an item
consumer thread id 89 consumes an item
consumer thread id 90 consumes an item
consumer thread id 91 consumes an item
consumer thread id 92 consumes an item
consumer thread id 93 consumes an item
consumer thread id 94 consumes an item
consumer thread id 95 consumes an item
consumer thread id 96 consumes an item
consumer thread id 97 consumes an item
consumer thread id 98 consumes an item
consumer thread id 99 consumes an item
All threads complete
bash-4.4$
```