Name:
Section:
University ID:

Lab 3 Report

Summary:

20pts

In this lab, we started out by practicing our programming using multiple threads. We started by creating new threads using the pthread_create() function, and joining them back together after they have terminated using the pthread_join() function. We then explored how to use pthread_mutex_lock and pthread_mutex_unlock to synchronize threads that share a common data source. We also learned about Conditional Variables and how they are used to block or unblock threads that are waiting for access to resources so they can finish their termination.

The biggest problem that I encountered during this lab was on the final part of the lab where you had to create a producer function to synchronously produce data for 100 different consumer threads. Initially, I had assumed that you were supposed to set producer_done = 1 every time you finished producing a new 10 supply. However, you only want to set producer_done = 1 when you are completely done producing.

Lab Questions:

3.1:

10pts 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, you can not see the threads outputs even after adding the sleep(5) to each of the thread functions. This is because when you call pthread_create() from the main thread (in the main method), the main thread returns from main() before the newly created threads have finished executing. If the main thread performs a return from main(), it causes the termination of all threads in the process.

5pts 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?

Yes, now I can see the outputs from both threads, followed by the output from main. This is because the pthread_join() function makes the calling thread wait for threads1 and threads2 to terminate, before finishing the rest of its execution.

5pts Include your commented code.

Lab3 > C ex1.c > thread2()

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

// function prototype for thread 1
void* thread1();
// function prototype for thread 2
void* thread2();


void main() {
    pthread_t i1;   //thread 1 ID
    pthread_t i2;   //thread 2 ID

    /* Create two threads */
    pthread_create(&i1, NULL, (void*)&thread1, NULL);   //thread1 starts execution on thread1()
    pthread_create(&i2, NULL, (void*)&thread2, NULL);   //thread2 starts execution on thread2()

    /* main waits for the two threads to finish*/
    pthread_join(i1, NULL);
    pthread_join(i2, NULL);

    /* main's print */
    printf("Hello from main");

}
/* Thread1 function */
void* thread1() {
    sleep(5);   //sleep for 5 milisec once thread created
    printf("Hello from thread1");
}

/* Thread2 function */
void* thread2() {
    sleep(5); //sleep for 5 milisec once thread created
    printf("Hello from thread2");
}
```

3.2:

3.2.1:

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

The output value for v=0

```
Lab3 > t1_out
1    v=0
```

15 pts 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 value of v returned is now v=-990. When removing the pthread_mutex_lock and pthread_mutex_unlock statements, both of the threads created by the main thread are running at the same time. Because they both manipulate the value of v, they are both trying to perform operations at the same time. So instead of incrementing up to 990, then back down to 0, the decrement function is executed at the same time as the increment function. This results in v=-990.

3.2.2:

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

```
70  /* third routine for thread three */
71  void* again(){
72      pthread_mutex_lock(&mutex);
73
74      /* again thread waits for done_1 ==1 */
75      while(done_1 == 0)
76      pthread_cond_wait(&done_world, &mutex); // listen for signal from world()
77
78      printf("Again!");    //print statment
79      fflush(stdout);     //flush buffer
80      pthread_mutex_unlock(&mutex);       //unlock mutex
81
82      return ;
83  }
```

```
1   Hello World Again!
```

3.3:

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

```c
/* Completed code for producer */
void *producer(void *arg)
{
  int producer_done = 0;

  while (!producer_done)
  {

    /* Wait for consumers to consume all of current supply */
    pthread_mutex_lock(&mut);
    while (supply > 0) {
      pthread_cond_wait(&producer_cv, &mut);  //wait for consumer threads to signal producer
    }

    /* If haven't used all of consumer threads, produce more supply.
    when no consumer threads left, exit producer*/
    if (num_cons_remaining != 0) {
      supply = NUM_ITEMS_PER_PRODUCE;   //produce 10 supply
      printf("produced 10 new supply\n");
      fflush(stdout);
    } else {
      producer_done = 1;  //flag to exit producer
      return NULL;
    }
    /* If supply has been produced, signal consumers to resume consumption */
    if (supply != 0) {
      pthread_cond_signal(&consumer_cv);
    }
    pthread_mutex_unlock(&mut);

  }
  return NULL;
}
```

```
consumer thread id 46 consumes an item
consumer thread id 47 consumes an item
consumer thread id 48 consumes an item
consumer thread id 49 consumes an item
produced 10 new supply
consumer thread id 50 consumes an item
consumer thread id 51 consumes an item
consumer thread id 52 consumes an item
consumer thread id 53 consumes an item
consumer thread id 54 consumes an item
consumer thread id 55 consumes an item
consumer thread id 56 consumes an item
consumer thread id 57 consumes an item
consumer thread id 58 consumes an item
consumer thread id 59 consumes an item
produced 10 new supply
consumer thread id 60 consumes an item
consumer thread id 61 consumes an item
consumer thread id 62 consumes an item
consumer thread id 63 consumes an item
consumer thread id 64 consumes an item
consumer thread id 65 consumes an item
consumer thread id 66 consumes an item
consumer thread id 67 consumes an item
consumer thread id 68 consumes an item
consumer thread id 69 consumes an item
produced 10 new supply
consumer thread id 70 consumes an item
consumer thread id 71 consumes an item
consumer thread id 72 consumes an item
consumer thread id 73 consumes an item
consumer thread id 74 consumes an item
consumer thread id 75 consumes an item
consumer thread id 76 consumes an item
consumer thread id 77 consumes an item
consumer thread id 78 consumes an item
consumer thread id 79 consumes an item
produced 10 new supply
consumer thread id 80 consumes an item
consumer thread id 81 consumes an item
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
produced 10 new supply
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$
```