Name:
Section:
University ID:

CprE 308 Lab 3 Report

Summary:

This lab focuses on parallel programming, introducing common concurrency problems and their solutions. This lab gives more experience manipulating mutexes and threads, further refining previous skills.

I have only a small amount of experience with c's threading, and as my focus is almost completely shot because of other classes, I appreciate such a well guided and clearly documented assignment *immensely*.

Lab Questions:

3.1:

10 pts Add a sleep(5) statement in the beginning of the thread functions. Compile and run the program. Can you see the threads' output? Why?

You cannot see the threads' output as the program is terminating before the threads complete, losing any output they may give.

5 pts Add two *pthread_join* calls for t1 and t2 just before the printf statement in main. Recompile and rerun the program. What is the output? Why?

Hello from thread 2
Hello from thread 1
Hello from the main thread

The process waits for the threads to complete before continuing, allowing their outputs to be printed before termination.

5 pts Include your code with comments explaining the usage of *pthread_create()* and *pthread_join()*.

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

void* thread1() {
   sleep(5);
   printf("Hello from thread 1\n");
   return NULL;
}
void* thread2() {
   sleep(5);
   printf("Hello from thread 2\n");
   return NULL;
}


int main (int argc, char *argv[])
{
   pthread_t t1, t2;

   //Create and launch a new thread, stored in t1 or t2 and executing
   //thread1 or thread2 respectively
   pthread_create(&t1, NULL, thread1, NULL );
   pthread_create(&t2, NULL, thread2, NULL );

   //Wait for the completion of t1 and t2, then continue
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   printf("Hello from the main thread\n");
}
```

3.2:

3.2.1:

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

v = 0

15 pts Delete the *pthread_mutex_lock* and *pthread_mutex_unlock* statements 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

The output is different because thread2 is overwriting the changes made by thread1, as the removal of the mutex locks allow both threads to act on the critical area at the same time.

3.2.2:

20 pts Modify the program by adding another thread called *tid_again* and a function called *again*. Create a second conditional variable *done_world* to synchronize the three threads so that they print out "Hello World Again!". Include your modified code with comments labeling what you added or changed.

```
/* 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 func

/* global variable shared by threads */
pthread_mutex_t mutex;                  // mutex
pthread_cond_t done_hello;   // conditional variable
pthread_cond_t done_world;  //----------------Added a second global conditional
int done = 0;               // testing variable

int main (){
   pthread_t tid_hello, tid_world; // thread id
   pthread_t tid_again;           //---------------------Added another thread

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

   //-----------------------------------------------Initialize added conditional
   pthread_cond_init(&done_world, NULL);


   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); //-------Create third thread

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

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

```
void* hello() {
   pthread_mutex_lock(&mutex);
   printf("Hello ");   //-----------------------------------Capitalized 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
}


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

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

   done = 2;       //--------------------------Update var for again()'s while

   printf("World");    //-----------------------------------Capitalized world
   fflush(stdout);
   pthread_cond_signal(&done_world);        //----------------Signal alone() thread
   pthread_mutex_unlock(&mutex); // unlocks mutex
}


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

   while(done < 2)
           pthread_cond_wait(&done_world, &mutex);

   printf(" Again!");
   fflush(stdout);
   pthread_mutex_unlock(&mutex); // unlocks mutex
}
```

3.3:

20pts Fill in the code for the producer and make the program run as described. Include your modified code with comments labeling what you added or changed.

```
/*
 * Fill in the "producer" function to satisfy the requirements
 * set forth in the lab description.
 */

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

/*
 * the total number of consumer threads created.
 * each consumer thread consumes one item
 */
#define TOTAL_CONSUMER_THREADS 100

/* This is the number of items produced by the producer each time. */
#define NUM_ITEMS_PER_PRODUCE  10

/*
 * the two functions for the producer and
 * the consumer, respectively
 */
void *producer(void *);
void *consumer(void *);


/********** global variables begin *******/

pthread_mutex_t mut;
pthread_cond_t producer_cv;
pthread_cond_t consumer_cv;
int supply = 0;  /* inventory remaining */

/*
 * Number of consumer threads that are yet to consume items.  Remember
 * that each consumer thread consumes only one item, so initially, this
 * is set to TOTAL_CONSUMER_THREADS
 */
int num_cons_remaining = TOTAL_CONSUMER_THREADS;

/************** global variables end ***************************/
```

```
int main(int argc, char * argv[])
{
  pthread_t prod_tid;
  pthread_t cons_tid[TOTAL_CONSUMER_THREADS];
  int thread_index[TOTAL_CONSUMER_THREADS];
  int i;

  /********** initialize mutex and condition variables ***********/
  pthread_mutex_init(&mut, NULL);
  pthread_cond_init(&producer_cv, NULL);
  pthread_cond_init(&consumer_cv, NULL);
  /*************************************************************/


  /* create producer thread */
  pthread_create(&prod_tid, NULL, producer, NULL);

  /* create consumer thread */
  for (i = 0; i < TOTAL_CONSUMER_THREADS; i++)
  {
          thread_index[i] = i;
    pthread_create(&cons_tid[i], NULL, consumer, (void *)&thread_index[i]);
  }

  /* join all threads */
  pthread_join(prod_tid, NULL);
  for (i = 0; i < TOTAL_CONSUMER_THREADS; i++)
    pthread_join(cons_tid[i], NULL);

  printf("All threads complete\n");

  return 0;
}




/***************** Consumers and Producers *****************/

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

  while (!producer_done)
  {
    pthread_mutex_lock(&mut);                   //Lock supply
```

```
    if(supply == 0){                        //If empty

      supply += NUM_ITEMS_PER_PRODUCE;          //Make more

      pthread_cond_signal(&consumer_cv);        //Spread the good word
      pthread_cond_wait(&producer_cv, &mut);    //Twiddle thumbs
    }

    pthread_mutex_unlock(&mut);                 //Unlock supply

    if(!num_cons_remaining)                     //If everyone died
      producer_done = 1;                        //Follow them

  }
  return NULL;
}



void *consumer(void *arg)
{
  int cid = *((int *)arg);

  pthread_mutex_lock(&mut);
  while (supply == 0)
    pthread_cond_wait(&consumer_cv, &mut);

  printf("consumer thread id %d consumes an item\n", cid);
  fflush(stdin);

  supply--;
  if (supply == 0)
    pthread_cond_broadcast(&producer_cv);

  num_cons_remaining--;

  pthread_mutex_unlock(&mut);

  return NULL;
}
```