Name: Mason Walls

Section: 7

University ID: 910333448

**CprE 308 Lab 3 Report**

**Summary (10 pts)**:

|  |
| --- |
| **During this lab, I learned a lot about pthreads and how they work. In the first part of the lab, I learned about the importance of pthread\_join, and how it may be required to actually capture thread output, especially if the main is very short. In parts 2 and 3, I learned why mutexes are important in threads. Because V was being editied by both threads, a mutex was used so only one could change it at a time. When I removed it, the data was accessed simultaneously which created an incorrect return. In part 3 I learned how to re-start a thread that had signaled another. This made it possible for a thread to run as many times as needed if others depended on it.** |
| **Overall, I learned a lot about how threads function. This includes how to properly wait for them, how to start and stop threads from one another, and how to re-open a thread that needs to run more than once.** |

**Lab Questions:**

**3.1:**

**10 pts** Add a sleep(5) statement in the beginning of both thread1 and thread2. Compile and run the program. Do the messages get printed? Why or why not?

|  |
| --- |
| The messages do not get printed. The two created threads get the sleep statement, but the main thread does not. The main thread is still able to get to its print and return statement before threads 1 and 2 get a chance to print. |

**5 pts** Add two pthread\_join() calls to join t1 and t2 just before the print statement in main(). Compile and run the program. Do the threads' messages get printed? Why or why not?

|  |
| --- |
| The threads messages do get printed. pthread\_join works like waitpid(). We specifically waited for the threads to terminate before proceeding, therefore we got the printout. |

**5 pts** Include your code with comments explaining the usage of *pthread\_create()* and *pthread\_join()*.

|  |
| --- |
| #include <stdio.h>  #include <pthread.h>  #include <unistd.h>  void \* thread1(void \* ptr);  void \* thread2(void \* ptr);  int main()  {  void\* val;  pthread\_t t1;  pthread\_t t2;  pthread\_create(&t1, NULL, thread1, NULL);//create thread using t1's address, and the function it will run  pthread\_create(&t2, NULL, thread2, NULL);//create thread using t2's address and the function it will run  pthread\_join(t1,&val);//join t1 and capture its return value  pthread\_join(t2, &val); //join t2 and capture its return value  printf("Hello from the main thread\n");  return 0;  }  void \* thread1(void \* ptr)  {  sleep(5);  printf("Hello from thread 1\n");  }  void \* thread2(void \* ptr)  {  sleep(5);  printf("Hello from thread 2\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 either the same as, or different than, before.

|  |
| --- |
| V = -990 This output happens because of the removal of the critical region lock. Unlocking the critical region means that the threads are updating values at the same time, and may not reflect the changes that one thread or the other are doing on the data. |

**3.2.2:**

**20 pts** 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 three routines called by threads  void\* world();  void\* again();  /\* global variable shared by threads \*/  pthread\_mutex\_t mutex; // mutex  pthread\_cond\_t done\_hello; // conditional variable  pthread\_cond\_t done\_world; //conditonal variable  int done = 0; // testing variable  int doneAgain = 0; //testing variable  int main (){  pthread\_t tid\_hello, tid\_world, tid\_again; // thread id    /\* initialize mutex and cond variable \*/  pthread\_mutex\_init(&mutex, NULL);  pthread\_cond\_init(&done\_hello, NULL);  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);  /\* main waits for the two threads to finish \*/  pthread\_join(tid\_hello, NULL);  pthread\_join(tid\_world, NULL);  pthread\_join(tid\_again, NULL);  printf("\n");  return 0;  }  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  }  void\* world() {  pthread\_mutex\_lock(&mutex);  /\* world thread waits until done == 1. \*/  while(done == 0)  pthread\_cond\_wait(&done\_hello, &mutex);  printf("world");  fflush(stdout);  doneAgain = 1;  pthread\_mutex\_unlock(&mutex); // unlocks mutex  }  void\* again() {  pthread\_mutex\_lock(&mutex);  /\* again thread waits until doneAgain == 1. \*/  while(doneAgain == 0)  pthread\_cond\_wait(&done\_world, &mutex);  printf(" again");  fflush(stdout);  done = 1;  pthread\_mutex\_unlock(&mutex); // unlocks mutex  } |

**3.3:**

**20pts** 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)  {  /\* TODO: fill in the code here \*/  pthread\_mutex\_lock(&mut);//lock mutex  supply = supply + NUM\_ITEMS\_PER\_PRODUCE;//create an initial supply  pthread\_cond\_broadcast(&consumer\_cv);//tell consumers we are ready  pthread\_mutex\_unlock(&mut);//unlock mutex  while (!producer\_done)//create second loop to keep production at a minimum  {  pthread\_mutex\_lock(&mut);//lock mutex  while(supply == 0)//wait for supply to be 0  pthread\_cond\_wait(&producer\_cv, &mut);//wait for a consumer to tell us to make more  supply = supply + NUM\_ITEMS\_PER\_PRODUCE;//increase supply by 10  pthread\_cond\_broadcast(&consumer\_cv);//tell consumers we are ready  pthread\_mutex\_unlock(&mut);//unlock mutex  if(num\_cons\_remaining == 0)//if there are no consumers left, close out of both loops  producer\_done = 1;  }  }  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(stdout);  supply--;  if (supply == 0)  {  pthread\_cond\_broadcast(&producer\_cv);  }  num\_cons\_remaining--;  pthread\_mutex\_unlock(&mut);  return NULL;  } |