Name: Shubham Sharma
Section: L
University ID: 866653811

Lab 3 Report

Summary:

This lab we experiment programming with multiple threads. We learn how the create, manage, and debug with multiple threads in C. Synchronization of threads with the help of mutex is also part of the lab, which helps in using a common data source.

Lab Questions:

3.1:

12pts 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, there is no tread output.
This is because we lack the function pthread_join for both of our treads. Due to the sleep in the beginning of both of our threads, the main method terminates before the thread's sleep is complete. Hence, the program is unable to print from the thread functions.

4pts 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?

The output of the program is as follows:

```
Main: Before Threads
Hello,I am thread 2
Hello,I am thread 1
Main: Threads complete
```

We can see the output of the threads this time, due to the addition of the line pthread_join for both the threads. Using the function pthread_join, it blocks the calling thread until the thread associated with it is complete.

4pts Include your commented code.

```c
// PTHREAD EXAMPLE [ex1.c]
// AUTHOR: Shubham Sharma, shubham@iastate.edu

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Header file for sleep()
#include <pthread.h> //Header file to link pthread_t

// Thread 1 function:
// Prints after 5 secs
void *thread1()
{
    sleep(5);
    printf("Hello,I am thread 1\n");
    return NULL;
}

// Thread 2 function:
// Prints after 5 secs
void *thread2()
{
    sleep(5);
    printf("Hello,I am thread 2\n");
    return NULL;
}

// Main function:
// Creates and runs 2 thread functions.
int main()
{
    pthread_t i1,i2;
    printf("Main: Before Threads\n");
    pthread_create(&i1, NULL, thread1, NULL);
    pthread_create(&i2, NULL, thread2, NULL);
    pthread_join(i1, NULL);
    pthread_join(i2, NULL);
    printf("Main: Threads complete\n");
    exit(0);
}
```

3.2:

3.2.1:

4pts Compile and run t1.c, what is the output value of v?

Output:
v=0

16pts 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.

Output (varies on every run):
v=-980
Explanation:

Without the pthread_mutex_lock and pthread_mutex_unlock statement in both increment and decrement threads, there would be no form of thread synchronization. Both the threads run simultaneously. And as both the threads depend on the same global variable it causes conflict and the program outputs a value that is not 0.

3.2.2:

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

```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();

/* global variable shared by threads */
pthread_mutex_t     mutex;          // mutex
pthread_cond_t      done_hello;     // conditional variable
pthread_cond_t    done_world;   // added conditional variable
int             done = 0;       // testing variable

int main (int argc, char *argv[]){
    pthread_t   tid_hello, // thread id
            tid_world, tid_again;
    /*  initialization on 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, (void*)&hello, NULL);
//thread creation
    pthread_create(&tid_world, NULL, (void*)&world, NULL);
//thread creation
    pthread_create(&tid_again, NULL, (void*)&again, NULL); //added
new thread creation statement

    /* main waits for the two threads to finish */
    pthread_join(tid_hello, NULL);
    pthread_join(tid_world, NULL);
    pthread_join(tid_again, NULL); //Add pthread_join join here

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

```c
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_hello, &mutex);

    printf("World ");
    fflush(stdout);
    done = 1;
    pthread_cond_signal(&done_world);    // Added signal world()
thread
    pthread_mutex_unlock(&mutex); // unlocks mutex

    return ;
}

// Added new function to print again in diff thread.
void again() {
    pthread_mutex_lock(&mutex);
    /* world thread waits until done == 1. */
    while(done == 0)
      pthread_cond_wait(&done_world, &mutex);

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

3.3:

40pts Include your modified code with your lab submission and comment on what you added or changed.

Portion of the output:

```
Produced: 10 more items
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
Produced: 10 more items
consumer thread id 98 consumes an item
consumer thread id 99 consumes an item
consumer thread id 21 consumes an item
consumer thread id 32 consumes an item
consumer thread id 5 consumes an item
consumer thread id 54 consumes an item
consumer thread id 65 consumes an item
consumer thread id 76 consumes an item
consumer thread id 86 consumes an item
consumer thread id 97 consumes an item
All threads complete
```

Code:

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

#include <pthread.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
```

```c
/* 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)
  {
    /* fill in the code here */

    //Locks for the while conditional loop
    pthread_mutex_lock(&mut);

    //Waits until the consumer consumes 10
    while (supply > 0)
      pthread_cond_wait(&producer_cv, &mut);

    //All consumer threads are complete
    if(num_cons_remaining == 0) return;
```

```
    // Prints 10 more items.
    printf("Produced: 10 more items\n");
    fflush(stdin);

    // Increments the supply chain
    supply += 10;
    // Notifies the consumer to work
    pthread_cond_broadcast(&consumer_cv);

    //Unlocks the mutex
    pthread_mutex_unlock(&mut);
  }
  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;
}
```