Name: Muhammad Rizwan Khalid

Class: BSCS-6A

Reg. No: 180459

CS 330: Operating Systems

Lab 2: Pthreads

Thread Creation:

1. To compile the pthread.c file, "cc -pthread pthread.c" command is used in linux terminal. It compiles and creates an executable file and issuing "./a.out" command ran the program in the file.
2. In the code we are creating two threads using pthread API for C/C++. pthread_create function creates and independent thread which will execute the print message function. This function returns the thread id. pthread_join waits for the termination of other thread.
3. In this program same function is used in each thread with different arguments. However, the functions of two threads can be different.
4. Overall in the code, the threads terminates by explicitly calling pthread_exit(), by letting the function return, or by a call to the function exit() which will terminate the process including any threads.

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

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     const char *message1 = "Thread 1";
     const char *message2 = "Thread 2";
     int iret1, iret2;

    /* Create independent threads each of which will execute function */

     iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
     if(iret1)
     {
         fprintf(stderr,"Error - pthread_create() return code: %d\n",iret1);
         exit(EXIT_FAILURE);
     }

     iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);
     if(iret2)
     {
         fprintf(stderr,"Error - pthread_create() return code: %d\n",iret2);
         exit(EXIT_FAILURE);
     }

     printf("pthread_create() for thread 1 returns: %d\n",iret1);
     printf("pthread_create() for thread 2 returns: %d\n",iret2);

     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     exit(EXIT_SUCCESS);
}

void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s \n", message);
}
```

Illustration 1: Code for Thread Creation

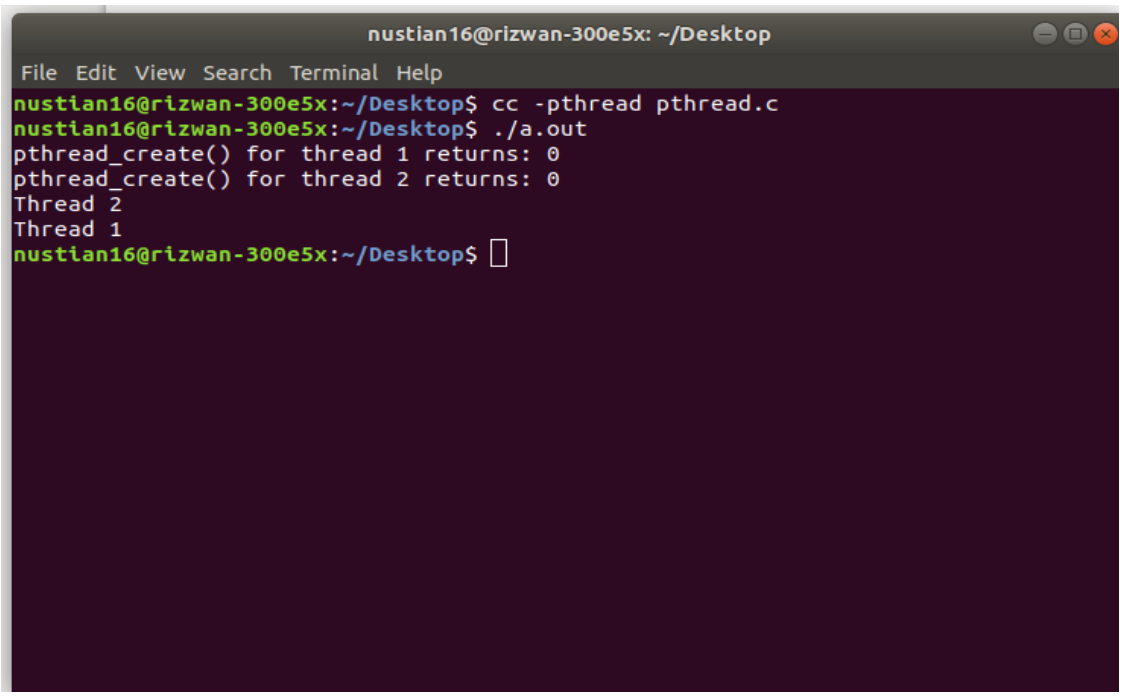

Illustration 2: Output of thread.c

Mutually Exclusive Locks (mutexes):

1. To compile the mutex.c file, "cc -pthread mutex.c" command is used in linux terminal. It compiles and creates an executable file and issuing "./a.out" command ran the program in the file.
2. In the code, we are creating two threads again by using the pthread_create function. However, we are passing different thread function this time. However, we are using the same function in both threads with the same arguments.
3. We use mutexes to prevent data inconsistencies by multiple threads doing operation upon same memory area at the same time.
4. When a mutex lock is attempted against a mutex which is held by another thread, the thread is blocked until the mutex is unlocked. When a thread terminates, the mutex does not unless explicitly unlocked.
5. Thread one accessed the counter variable and incremented it while second thread gets blocked because of mutex and after unlocking, second thread accessed the modified value of the variable.

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

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     const char *message1 = "Thread 1";
     const char *message2 = "Thread 2";
     int iret1, iret2;

    /* Create independent threads each of which will execute function */

     iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
     if(iret1)
     {
         fprintf(stderr,"Error - pthread_create() return code: %d\n",iret1);
         exit(EXIT_FAILURE);
     }

     iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);
     if(iret2)
     {
         fprintf(stderr,"Error - pthread_create() return code: %d\n",iret2);
         exit(EXIT_FAILURE);
     }

     printf("pthread_create() for thread 1 returns: %d\n",iret1);
     printf("pthread_create() for thread 2 returns: %d\n",iret2);

     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     exit(EXIT_SUCCESS);
}

void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s \n", message);
}
```

Illustration 3: Code for Mutually Exclusive Lock

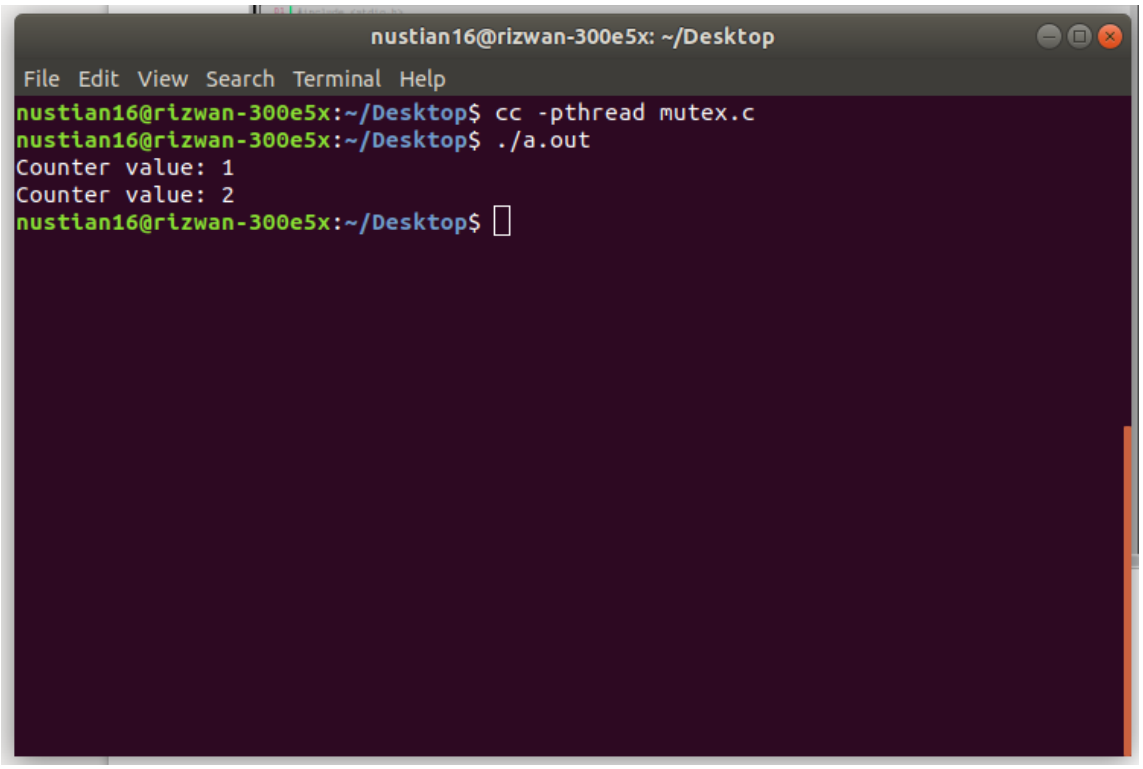

Illustration 4: Output of mutex.c

Thread Joins:

1. To compile the join.c file, "cc -pthread join.c" command is used in linux terminal. It compiles and creates an executable file and issuing "./a.out" command ran the program in the file.
2. In this code we are creating ten threads, and passing same function to all the threads having same argument null. A join is performed when one wants to wait for a thread to finish. A thread calling routine may launch multiple threads then wait for them to finish to get the results. One waits for the completion of the threads with a join.
3. Every thread is accessing the same variable and modifying the received version of the variable. That's why we have also used the mutexes because threads are accessing the same memory area.
4. In the second loop join is wait for every thread to complete and after that we are printing the final value of the counter variable.

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

#define NTHREADS 10
void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

main()
{
   pthread_t thread_id[NTHREADS];
   int i, j;

   for(i=0; i < NTHREADS; i++)
   {
      pthread_create( &thread_id[i], NULL, thread_function, NULL );
   }

   for(j=0; j < NTHREADS; j++)
   {
      pthread_join( thread_id[j], NULL);
   }

   /* Now that all threads are complete I can print the final result.     */
   /* Without the join I could be printing a value before all the threads */
   /* have been completed.                                                */

   printf("Final counter value: %d\n", counter);
}

void *thread_function(void *dummyPtr)
{
   printf("Thread number %ld\n", pthread_self());
   pthread_mutex_lock( &mutex1 );
   counter++;
   pthread_mutex_unlock( &mutex1 );
}
```

Illustration 5: Code for Thread Joins

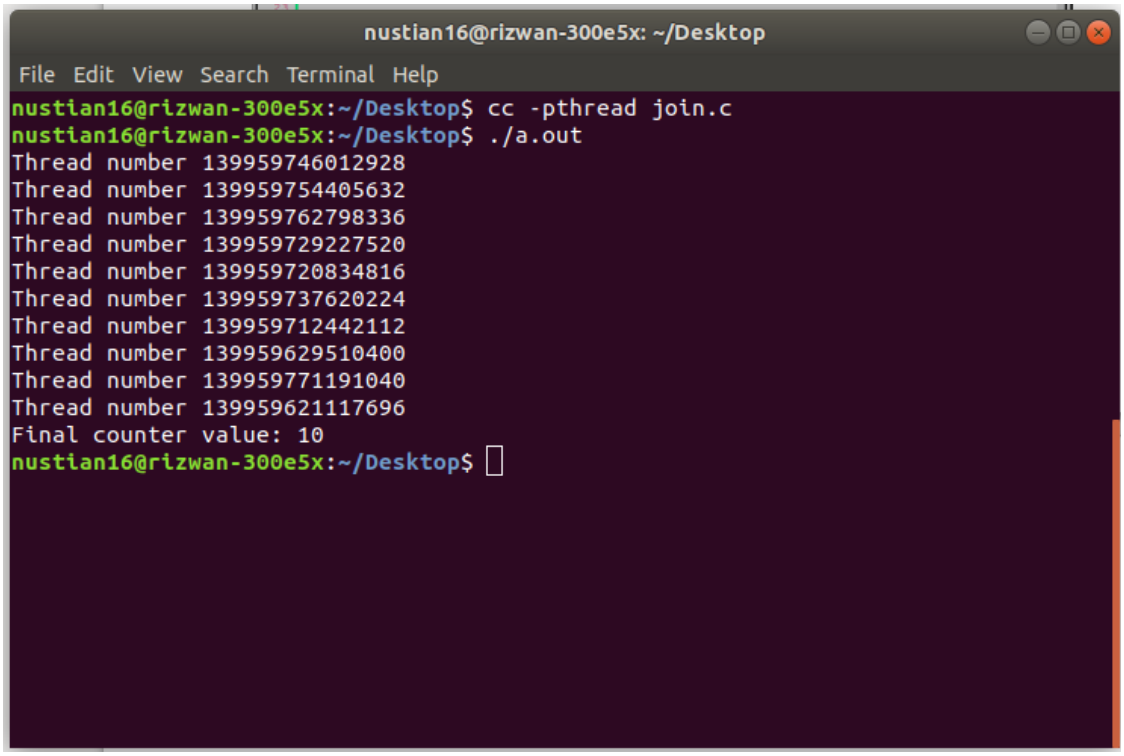

Illustration 6: Output of join.c