NATIONAL UNIVERSITY
OF COMPUTER & EMERGING SCIENCES
PESHAWAR CAMPUS

Problem Set:	Assignment 05	**Semester**:	Spring 2013
Points:	4		
Date Set:	April 3, 2013	**Due Date**:	April 10, 2013
Course:	CS206 Operating Systems	**Instructor**:	Nauman

Note: Code in the following is intentionally left low-quality. Please write the code yourself.

1. The following code deals with basic thread creation.

   (a) Write the following program and observe and *explain* the output.

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

void * thread1()
{
  int c = 0;
    while(c++ < 100)
      printf("Hello!!\n");
}

void * thread2()
{
  int c = 0;
    while(c++ < 100)
      printf("How are you?\n");
}

int main()
{
        int status;
        pthread_t tid1,tid2;

        pthread_create(&tid1,NULL,thread1,NULL);
        pthread_create(&tid2,NULL,thread2,NULL);
        pthread_join(tid1,NULL);
        pthread_join(tid2,NULL);
        return 0;
}
```

   (b) Modify the program to create four threads using the same two functions (`thread1` and `thread2`)

   (c) Run both versions and include screenshots of the output.

   Note To compile the program, you need to include the following switch in the compilation command: `-lpthread`. This tells gcc to use the registered pthread library.

2. The following code demonstrates the use of semaphores. It is incomplete but in a compilable form.

```c
/* Includes */
#include <unistd.h>     /* Symbolic Constants */
#include <sys/types.h>  /* Primitive System Data Types */
#include <errno.h>      /* Errors */
#include <stdio.h>      /* Input/Output */
#include <stdlib.h>     /* General Utilities */
#include <pthread.h>    /* POSIX Threads */
#include <string.h>     /* String handling */
#include <semaphore.h>  /* Semaphore */

/* prototype for thread routine */
void handler ( void *ptr );

/* global vars */
/* semaphores are declared global so they can be accessed
   in main() and in thread routine,
   here, the semaphore is used as a mutex */
sem_t mutex;
int counter; /* shared variable */

```

```c
int main()
{
    int i[2];
    pthread_t thread_a;
    pthread_t thread_b;

    i[0] = 0; /* argument to threads */
    i[1] = 1;

    sem_init(&mutex, 0, 1);      /* initialize mutex to 1 - binary semaphore */
                                 /* second param = 0 - semaphore is local */

    pthread_create (&thread_a, NULL, (void *) &handler, (void *) &i[0]);
    pthread_create (&thread_b, NULL, (void *) &handler, (void *) &i[1]);

    pthread_join(thread_a, NULL);
    pthread_join(thread_b, NULL);

    sem_destroy(&mutex); /* destroy semaphore */

    exit(0);
}
```

```c
void handler ( void *ptr )
{
  int iter = 0;
  int x;
  x = *((int *) ptr);
  while(iter++ < 100000) {
    // printf("Thread %d: Counter Value: %d\n", x, counter);
    // printf("Thread %d: Incrementing Counter...\n", x);
    counter++;
    // printf("Thread %d: New Counter Value: %d\n", x, counter);
  }
  printf("Thread %d, counter at end = %d \n", x, counter);
  pthread_exit(0); /* exit thread */
}
```

(a) Run the program and observe the output. Explain what the expected output is and what output is being produced by the program.

(b) Modify the program to use locking using the declared semaphore. The functions corresponding to down and up operations are:

down : `sem_wait(&mutex)`

up : `sem_post(&mutex)`

(c) Recompile the program after making changes and demonstrate the correct output.

(d) Include screenshots of runs of both versions.