University of Wolverhampton
School of Mathematics and Computer Science

6CS005 High Performance Computing Week 6 Workshop

Tasks – More OpenMP Multithreading

You may need to refer to the lecture slides in order to complete these tasks.

1. The following program uses semaphores to manage the loan of 3 books to a group of 5 borrowers:

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

sem_t semaBooks;
int booksAvailable = 3;

void *borrower(void *p)
{
  for(int i = 0; i<=10; i++){
    printf("Borrower %d wants to borrow a book. ",(int)p);
    printf("Books available = %d\n",booksAvailable);
    sem_wait(&semaBooks); //Borrowing a book
    printf("Borrower %d gets a book. ", (int)p);
    booksAvailable--;
    printf("Books available = %d\n",booksAvailable);
    usleep(10000); //reading book
    printf("Borrower %d is returning a book. ", (int)p);
    booksAvailable++;
    printf("Books available = %d\n",booksAvailable);
    sem_post(&semaBooks); //Returning a book
    usleep(10000);
  }
  return NULL;
}

void main()
{
  pthread_t thrID1, thrID2, thrID3;
  pthread_t thrID4, thrID5, thrID6;

  sem_init(&semaBooks, 0, 3 );
  pthread_create(&thrID1, NULL, borrower, (void *)1);
  pthread_create(&thrID2, NULL, borrower, (void *)2);
  pthread_create(&thrID3, NULL, borrower, (void *)3);
  pthread_create(&thrID4, NULL, borrower, (void *)4);
  pthread_create(&thrID5, NULL, borrower, (void *)5);
  pthread_create(&thrID6, NULL, borrower, (void *)6);
  pthread_join(thrID1, NULL);
  pthread_join(thrID2, NULL);
  pthread_join(thrID3, NULL);
  pthread_join(thrID4, NULL);
  pthread_join(thrID5, NULL);
  pthread_join(thrID6, NULL);
  sem_destroy(&semaBooks);
}.
```

   a. Convert it to use OpenMP locks.
   b. Modify the program so that it doesn't use locks.

2. The following program encodes 3 lower case letters into a numeric code:

```
#include <stdio.h>

long encode(char *s)
{
  long a,b,c,x;
  a = s[0];
  b = s[1];
  c = s[2];
  x = ((((a*69)+c)*137)+b)*39;
  x = x % 54321;
  return x;
}

void main()
{
  char s[100];
  long x;
  printf("Enter 3 lowercase letters: ");
  scanf("%s",s);
  s[3]='\0';
  x=encode(s);
  printf("Code for %s is %ld\n",s,x);
}
```

Enter it as "encode.c", build and run it and then enter 3 lower case letters. Note down the code it produces

The next program attempts to decode the code back to the original 3 letters.:

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

long encode(char *s)
{
  long a,b,c,x;
  a = s[0];
  b = s[1];
  c = s[2];
  x = ((((a*69)+c)*137)+b)*39;
  x = x % 54321;
  return x;
}

void main()
{
  char s[4];
  long x,y;
  int i,j,k;
  printf("Enter the code: ");
  scanf("%ld",&x);
  s[3]='\0';
  for(i=0;i<26;i++){
    s[0]=i+'a';
    for(j=0;j<26;j++){
      s[1]=j+'a';
      for(k=0;k<26;k++){
        s[2]=k+'a';
        y=encode(s);
        if(x==y){
          printf("The letters for code %ld are %s\n",y,s);
          exit(0);
        }
      }
    }
  }
}
```

a. Enter it as "decode.c", build and run it and verify that it decodes the letters successfully from the numeric generated by the previous program.
b. Modify the program to use the OpenMP for loop "collapse" method to spread the work across 16 threads.