BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI, K. K. BIRLA GOA CAMPUS

I SEMESTER 2019-2020

Operating Systems (CS F372)

Assignment 4 [Non-evaluative Assignment]

Resources

There are several resources you can access to make your life a little bit easier.

- man -k pthread is a good place to start.
- An Introduction to Programming with Threads. by A.D. Birrell. Has some nice, practical information.
- **Getting Started With POSIX Threads by Tom Wagner and Don Towsley [Another link for the same material is available here].**

Question #1.

Matrix Multiplication problem

Given two matrices ***A****m*×*k* (*m* rows and *k* columns) and ***B****k*×*n* (*k* rows and *n* columns), we want to compute the product of ***A*** and ***B*** into a matrix ***C*** of *m* rows and *n* columns. The entries of C on row *i* and column *j* is the sum of the products of the corresponding elements on row *i* of matrix ***A*** and column *j* of matrix ***B***.

Problem:

Write a program that reads in a *m*×*k* array and a *k*×*n* array in the main thread. Create a thread (say sub thread) which will create *m*×*n* threads, one for each entry *Ci*,*j* to compute its value. Each thread created must return the value back to the sub thread program and finally the result is to be printed in the main thread. Note that you should aim for maximum parallelism. More precisely, all created threads should run concurrently.

Question #2.
Write a program to demonstrate multiplication of two N digit numbers.
The computation for each digit in the result must be performed by one thread. The final generation of result (along with carry forwarding) will be taken care by the main thread.
Example 1:

```
                1      2      3      X
                       3      1
                -------------------
                1   |  2   |  3
         3  |   6   |   9
   ==========================
         3      7      (11)   3 (returned by threads t1 to t4)
= 3 8 1 3 (computed by main)
```

Example 2:

```
                1      2      3      X

                4      3      1

               ===============

                1   |   2    |  3

        3   |  6    |   9

   4   |  8   | 12

===================================

   4   |  11  | 19   |  11    |  3    (returned by threads t1 to t5)

===================================

   5   |   3  |    0 |     1  |  3    (computed by main)
```

Question #3.

Merging 2 arrays using threads

Suppose we have two sorted arrays A**[ ]** and **B[ ]** of **N** and **M** elements respectively. For simplicity, assume that all of these **N + M** elements are distinct. Merge these two arrays into a sorted one by using threads.

Take an element from array A, say A**[i],** which is larger than **i-1** elements of array **A**. To know the exact location of A[i] in merged array, one should find how many elements of array **B** is smaller than A**[i]**. With a slightly modified binary search, one can easily determine the location of A**[i]** in merged array. There are only three possibilities:

1. **A[i]** is less than **B[0]**: In this case, **A[i]** is larger than i elements in A and smaller than all elements in **B**. Therefore, **A[i]** should be in position **i** of the merged array.
2. **A[i]** is larger than **B[m-1]**: In this case, **A[i]** is larger than **i** elements in **A** and **m** elements in **B**. Therefore, **A[i]** should be in position **i+m** of the merged array.
3. **A[i]** is between **B[k-1]** and **B[k]**. In this case, **A[i]** is larger than **i** elements in **A** and **k** elements in **B**. Therefore, **A[i]** should be in position **i+k** of the sorted array.

The main thread creates a new thread to read **N** and **M**. Based on **N** and **M** values, the new thread allocates space for array **A**, array **B** and array **Result**. Then the new thread takes elements of array **A** and elements of array **B** from the user (You are allowed to use global variables in this program). After the successful execution of new thread, it joins the main thread. Then the main thread creates **N + M** concurrently executing threads, each of which handles an element in **A** or in **B**. Each of these threads determines its position in the merged array and writes the values into the corresponding location. After the completion of all the thread executions, the array **Result** will have a merged result. Thus, we use 2N (if N = M) threads, each of which takes $O(\log_2(N))$ comparisons to get the job done. The main thread should create a new thread to print the merged sorted resultant array **Result**.