COEN 177: Operating Systems
Lab assignment 4: Developing multi-threaded applications

Objectives

1. To develop multi-threaded applications
2. To demonstrate the use of pthreads for matrix multiplication

Guidelines

As noted in the class textbook[1], the increasing importance of parallel processing has led to the development of lightweight user-level multithreading libraries. Even so, there is still debate over whether multithreading is a better programming model than multiprocessing or other alternatives. Several prominent OS researchers have argued that normal programmers should almost never use threads because (a) it is too difficult to write logically correct multithreaded programs and (b) most of the common applications for multithreading can be accomplished in other, safer ways. These are important arguments to understand, regardless of whether you agree or disagree with them, as they illustrate pitfalls of multithreading that are important to avoid. The most common pitfall appears during concurrent access to shared memory. When multiple threads concurrently access shared memory, program behavior becomes undefined (unless all such accesses are read-only). This is because the thread schedule on the CPU is non-deterministic, so if the program result depends on the thread schedule, it may change unpredictably when you rerun the program. This behavior was the basis for some parts of Labs 2 and 3.

To ensure deterministic behavior of programs in which threads must access shared memory, some sort of synchronization mechanism must be implemented. Consequently,

1. The program behavior will depend only on a specific function of its input, not which thread runs first on the CPU, which can be considered random.
2. The program behavior will be deterministic and will not vary inappropriately from run to run.

This lab is designed to give you your first hands-on programming experience with developing multi-threaded applications.

C Program with threads (problems 1, 2, 7, and 8 of the class textbook)

In chapter 4 of the class textbook, the threadHello.c program has been discussed. The threads run on the CPU in different orders. Demonstrate each of the following steps to the TA to get a grade on this part of the lab assignment.

Step 1. Download the slightly revised threadHello.c program from Camino, then compile and run several times. The comment at the top of program explains how to compile and run the program.

Explain what happens when you run the threadHello.c program? Do you get the same result if you run it multiple times? What do you would happen if you were also running some other demanding processes (e.g., compiling a big program, playing a Flash game on a website, or watching streaming video) when you ran this program?

The function go() has the parameter arg passed as a local variable. Are these variables per-thread or shared state? Where does the compiler store these variables' states? Note that each thread has its own stack.

The main() function has a local variable i. Is this variable per-thread or shared state? Where does the compiler store this variable?

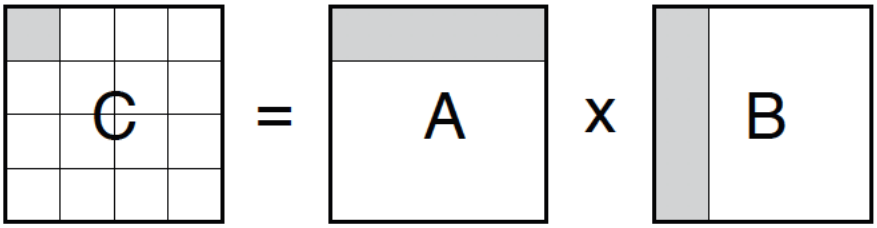
Step 2. Delete the second for loop in threadHello.c program so that the main function simply creates NTHREADS threads and then prints "Main thread done" and immediately returns. What are the possible outputs of the program now? Explain. Count the number of lines of output if needed.

Matrix multiplication with threads (problem 5 of the class text book)

Step 3. Write a program that uses threads to perform a parallel matrix multiply operation. To multiply two matrices, A and B, the result matrix C will entries $C_{i,j}$ is computed by taking the dot product of the i^{th} row of A and the j^{th} column of B: $C_{i,j} = \text{SUM}\ A_{i,k} * B_{k,j}$ for k = 0 to N-1, where N is the matrix size. We can

[1] J. Anderson and M. Dahlin, Operating Systems – Principles and Practice, Recursive Books, 2nd Edition, 2014

divide the work by creating one thread foreach row (or even each value) of C and executing those threads in parallel on a multiprocessor system. As shown in the following figure, each cell of the product matrix is the sum of products of row elements of A by the column elements of B.



You may fill in the entries of matrices A and B (`int matrixA[N][M], matrixB[M][L]`) using a random number generator as below:

```
srand(time(NULL));
for (int i = 0; i < N; i++)
     for (int j = 0; j < M; j++)
        matrixA[i][j] = rand() % 10;

for (int i = 0; i < M; i++)
     for (int j = 0; j < L; j++)
        matrixB[i][j] = rand() % 10;
```

Notes:

- The number of columns in the first matrix must equal to the number of rows in the second matrix.
- The values of N, M, and L should be large to exploit parallelism (e.g. N, M, L = 256).
- The output matrix would be defined `int matrixC[N][L];`
- Matrix multiplication is a loosely coupled decomposable problem--each dot product is independent.
- The number of threads to be created in the program is N, each thread `i` perfoms the following task:

```
for (int j = 0; j < L; j++){
     int temp = 0;
     for (int k = 0; k < M; k++){
         temp += matrixA[i][k] * matrixB[k][j];
     }
     matrixC[i][j] = temp;
 }
```

- The main thread needs to wait for all other threads to complete before it displays the resulting `matrixC`.

Requirements to complete the lab

1. Show the TA correct execution of the C programs.
2. Submit your answers to questions, observations, and notes as .txt file and upload to Camino
3. Submit the source code for all your programs as .c file(s) and upload to Camino.

Be sure to retain copies of your .c and .txt files. You will want these for study purposes and to resolve any grading questions (should they arise)

Please start each program/text file with a descriptive block that includes at minimum the following information:

```
//Name:
//Date:
//Title: Lab4 –
//Description:
```