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

Objectives

1. To develop multi-threaded application programs.
2. To demonstrate the use of threads in matrix multiplication.

Guidelines

As noted in the course textbook[1], parallel processing is increasingly important, which led to the development of lightweight user-level thread implementations. Even so, whether threads are a better programming model than processes or other alternatives remains open. Several prominent operating systems researchers have argued that normal programmers should rarely use threads because (a) it is just too hard to write multi-threaded programs that are correct and (b) most things that threads are commonly used for can be accomplished in other, safer ways. These are important arguments to understand—even if you agree or disagree with them, researchers point out pitfalls with using threads that are important to avoid. The most important pitfall is the concurrent access of threads to shared memory. Program behavior is undefined when threads concurrently read/write to shared memory. This is because the thread schedule on the CPU is non-deterministic. The program behavior completely changes when you rerun the program. This should have been obvious to you by now with the examples you have run in Lab2.

A synchronization mechanism needs to be implemented to ensure a deterministic behavior of programs where threads cooperate to access shared memory. Consequently,

1. The program behavior will be related to a specific function of input, not to the sequence of which thread runs first on the CPU
2. The program behavior will be deterministic and will not vary from run to run
3. These facts should not be IGNORED. Otherwise, the compiler will mess up the results and will not be according to what is thought would happen.

This lab will give you the first hands-on programming experience developing multi-threaded applications. In the coming labs, you will implement synchronization API primitives such as lock, semaphores, and condition variables.

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

In chapter 4 of the course textbook, the thread hello program has been discussed, and you have implemented this program in Lab3. You also realized the program behavior when you ran it multiple times, especially when your program runs on a computer that is busy running other demanding processes (e.g., compiling a big program, playing a Flash game on a website, or watching the streaming video).

Step 1. [10 points] Compile and run the modified program several times. How many threads complete their go function? Explain the reasoning behind the program's behavior.

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

void *go(void *);
#define NTHREADS 20
pthread_t threads[NTHREADS];
int main() {
    int i;
```

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

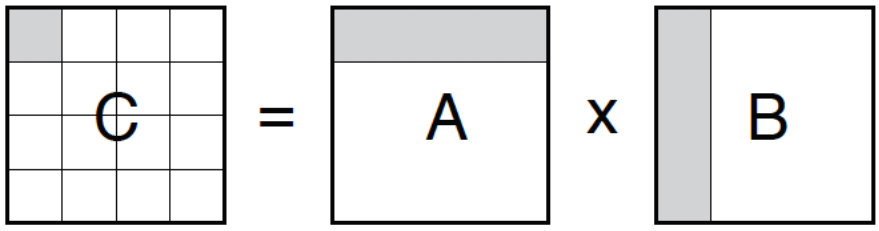
```
    for (i = 0; i < NTHREADS; i++)
        pthread_create(&threads[i], NULL, go, (void *)(size_t)i);

    printf("Main thread done.\n");
    return 0;
}
void *go(void *arg) {
 printf("Hello from thread %d with iteration %d\n",  (int)(unsigned
long)pthread_self(), (int)(unsigned long)(size_t *)arg);
 sleep(15);
 pthread_exit(0);
}
```

Step 2. [10 points] The function go() in the program in Step 1 has the parameter arg passed a local variable. Is the variable per thread variable or a shared state? Where does the compiler store the variable's state?

Step 3. [10 points] The main() in the program in Step 1 has local variable i. Is this variable per thread variable or a shared state? Where does the compiler store this variable?

Matrix multiplication with threads (problem 5 of the course textbook)

To multiply two matrices, C = A*B, the result entry $C_{(i,j)}$ 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 divide the work by creating one thread to compute each value (or each row) in C, and then executing those threads on different processors in parallel on multi-processor systems. As shown in the following figure, each is cell in the resulting matrix is the sum of the multiplication of row elements of the first matrix by the column elements of the second matrix.



You may fill in the entries of A and B matrices (`double 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();

srand(time(NULL));
for (int i = 0; i < M; i++)
     for (int j = 0; j < L; j++)
        matrixB[i][j] = rand();
```

The following are important notes:

- The number of columns in the first matrix must equal the number of rows in the second matrix.
- N, M, and L values must be large to exploit parallelism. Get the values via the arguments of main()
- The output matrix would be defined `double matrixC[N][L];`
- The Matrix multiplication is a loosely coupled problem and so decomposable, i.e., multiplication of each row of matrixA with all columns of the matrix can be performed independently and in parallel

- The number of threads to be created in the program equals N and each thread `i` would be performing the following task:

```
for (int j = 0; j < L; j++){
     double 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 before it displays the resulting `matrixC`

Step 4. [40 points] Write a program that implements matrix multiplication in a multi-threaded environment. You may use the following definitions and function prototypes:

```
//N threads
pthread_t threads[N];

//A, B, C matrices
double **matrixA, **matrixB, **matrixC;

//function prototypes
int main(int argc,char *argv[]) //read N, M, and L as command-line arguments
void initializeMatrix(int r,int c, double **matrix); //initialize matrix with
random values
void printMatrix(int r,int c, double **matrix); //print matrix
void *multiplyRow(void* arg) //thread multiply function

//creating N threads, each multiplying ith row of matrixA by each column of
matrixB to produce the row of matrixC
for(i=0;i<N;i++)
         pthread_create(&threads[i],NULL,multiplyRow,(void*)(size_t)i);
```

Step 5. [30 points] Modify your program in Step 4 to create N*L threads, each computing i^{th} row multiplied by j^{th} column.

Requirements to complete the lab

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

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

```
# Name: <your name>
# Date: <date> (the day you have lab)
# Title: Lab4 – task
# Description: This program computes … <you should
# complete an appropriate description here.>
```