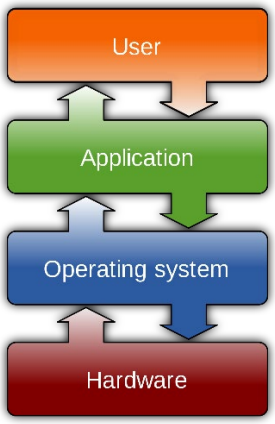
# Operating Systems
# Lab assignment 5: Developing multi-threaded applications




## Objectives

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

## Guidelines

As noted in the class text there is an increasing importance of parallel processing that led to the development of lightweight user-level thread implementations. Even so, the topic of 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 almost never 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. When threads concurrently read/write to shared memory, program behavior is undefined. This is because thread scheduling on the CPU is non-deterministic. The program behavior can completely change when you rerun the program.

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

1. The program behavior will be related to a specific function of input, not of 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 what is thought would happen.

This lab is designed to give you the first hands-on programming experience on developing multi-threaded applications. In this and the next upcoming lab, you will have a chance to work with Pthreads and develop a multi-threaded application.

## C Program with threads

In the text, the thread API has been discussed. It has been noted that 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.

## pthread_create()

Step 1. Compile and run the following program, then #include <stdio.h>

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

void *go(void *);
#define NTHREADS 10
pthread_t threads[NTHREADS];
int main() {
   int i;
   for (i = 0; i < NTHREADS; i++)
      pthread_create(&threads[i], NULL, go, &i);
   for (i = 0; i < NTHREADS; i++) {
    printf("Thread %d returned\n", i);
      pthread_join(threads[i],NULL);
   }
   printf("Main thread done.\n");
   return 0;
}
void *go(void *arg) {
 printf("Hello from thread %d with iteration %d\n",  (int)pthread_self(), *(int *)arg);
 return 0;
}
```

Explain what happens when you run this threadHello.c program; can you list how many threads are created and what values of i are passed? Do you get the same result if you run it multiple times? What if you are 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 run this program?

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

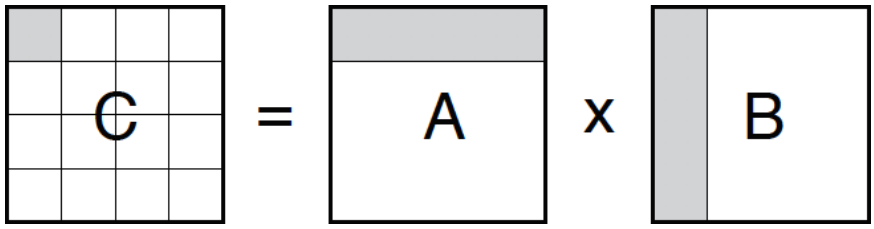
The main() has local variable i. Is this variable per-thread or shared state? Where does the compiler store this variable?

Write down your observations. You probably have seen that there is a bug in the program, where threads may print same values of i. Why?

Step 2. Write the fix for the program in Step 1, then run your revised program to demonstrate your fix.

## Matrix multiplication with threads

Step 3. Write a program that uses threads to perform a parallel matrix multiply. 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; j < N; i++)
     for (int j = 0; j < M; j++)
        matrixA[i][j] = rand();

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

The following are important notes:

- The number of columns of the first matrix must be equal to the number of rows in the second matrix.
- The values of N, M, and L must be large to exploit parallelism (e.g. N, M, L = 1024).
- 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 matrix can be performed independently and in parallel
- The number of threads to be created in the program equals to 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`

### Requirements to complete the lab

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

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 with a descriptive block that includes minimally the following information:

```
# Name: <your name>
```

```
# Date: <date> (the day you have lab)
```

```
# Title: Lab5 – task
# Description: This program computes … <you should
# complete an appropriate description here.>
```