# CSCI4730/6730 – Operating Systems

# Project #1: Multi-process and IPC

**Due date: 11:59pm, 9/19/2022**

## Description

In this project, you will design and implement a multi-process matrix multiplication program. The single-process version is provided, and you will expand it into the multi-process architectures.

## Part 0: Download and prepare the project file

You can download the project file (OS_Proj1.tar.gz) from ELC. You need to copy that into "odin" server using any ssh or sftp program.

- Once you have the tarball in odin server, you can unzip it with the following command:

```
% tar xzvf OS_Proj1.tar.gz
```

- Compile the program with "make" command.

```
% cd OS_Proj1
% make
```

## Part 1: Multi-process Matrix Multiplication Program (40%)

The main problem of a single-process program is the scalability.
To address the problem, you will convert the matrix multiplication program into the multi-process model. The main process creates multiple child processes and effectively divides work between child processes. The child process sends the result to the main process via Inter-process communication channel. We will use **pipe** in this project. The main process waits children processes and reads the result via IPC channel, and prints out the result on the screen.

- You will modify "matmul.c" to support a multi-process model.
- "matmul_core.c" contains for initialization, file read, and other utility functions. You can use any functions defined here, but you do NOT need to modify them.
- The program receives an input file name through the command-line argument.
  - (required) 1st argument: input file that contains two matrices
  - (optional) 2nd argument: crash rate (default is 0%)
  - Example: *./matmul ./inputs/mat1.txt*

- The input file contains
  - [value of m] [value of n] [value p]
  - Matrix A
  - Matrix B
  - For instance, mat1.txt file contains A[4*3] and B[3*4], the input file looks like the following.

```
sh-4.2$ cat matrices/mat1.txt
4 3 4

1 2 3
4 5 6
7 8 9
1 2 3

9 8 7 6
5 4 3 2
1 9 8 7
```

  - You can create input files manually to test the correctness of your implementation, or you can randomly generate input files with **`./matgen`** program.
    - ./matgen <m> <n> <p> → generates m*n and n*p matrices
    - It randomly generates two arrays and print it out into the screen. You can store it into a file as follows:
      - ./matgen 4 3 4 > ./matrices/test.txt

- We multiply two matrices, A and B, and store the result into a matrix C
  - Size of A: m * n
  - Size of B: n * p
  - Size of C: m * p
  - Wikipedia - Matrix multiplication:

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.

- "matmul.c" program has three parts:
    - **Part 1:** Allocate and initialize matrix data structures, and read the input file. This part is already implemented in the `main()` function. We use five matrices:
        - Input matrices: A[m][n], B[n][p]
        - Transposed matrix: B_tran[p][n]
            - For easier calculation, we use a transposed matrix of B, which flips a matrix over its diagonal.
        - Output matrices:
            - C_serial[m][p]: the result of the serial multiplication.
            - C_parallel[m][p]: the result of the parallel multiplication.
    - **Part 2:** Serial matrix multiplication part is already implemented in the `serial_mat_mult()` function.
    - **Part 3:** You will need to implement `parallel_mat_mult()` function.
        - The main process creates "m" processes and effectively divide the job into "m" children.
        - The main body of the child process is `child_process_core()` function, and you need to implement it.
        - After create the child processes, the main process need to wait for them and receive the results from each child via pipe. The final results need to be stored in "C_parallel".
    - At the end of the program, we compare the "C_serial" and "C_parallel", and they should be matched.

## Part 2: Recovery from the process crash (50%)

If one or more child processes crash before they complete the job (before sending the result through pipe), the final output will be incorrect. In this project, we will monitor the exit status of each child. If there is one or more child processes crashed, the main process will create new child processes to complete the job.

The parent process should "wait" for all child processes and monitor their exit status (hint: use waitpid()). If an exit status of a child process is abnormal termination by a signal, the parent process creates a new child process to re-do the incomplete job.

- You can use 2nd command-line arguments (integer between 0 and 30) to trigger crash. 30 means each child process has 30% chance to be killed abnormally by signal. 0 means 0% chance to crash.
- You can use `WIFEXITED` and `WIFSIGNALED` macros to identify the exit status of each child. You can find more details from the linux manual page. (% man 2 wait)
- We do NOT consider the crash of the parent.
- Any child process can crash if the given crash rate is > 0. Your design should be able to handle the recursive crashes. For instance, consider the following case:
    - The parent creates 10 children processes. 4 of them crash.

- The parent creates 4 additional processes to cover the crashed processes. But 2 of them are crashed.
- The parent creates 2 more, but 1 of them crashes.
- The parent creates 1 more child process, and it finishes jobs.

## Part 3: Project Report (10%)

Explain your program structure and IPC in README.pdf file. **Only "pdf" format** will be accepted. You do not need to explain the line by line of your code.
Please explain the high-level design of your multi-process structure and IPC.
And explain how your program handles crash.

## Submission

Submit a tarball file using the following command
%tar czvf p1.tar.gz README.pdf Makefile matmul.c

1. README.pdf file with:
    a. Your name
    b. Explain your design of multi-process structure and IPC.
    c. Explain how your program handles crash.
2. **Your code should be compiled and run correctly in odin machine.**
    a. TA will evaluate your code only on odin machine.
    b. If your code failed to compile on odin, you will get 0 point.
3. Submit a tarball through ELC.