

Operating Systems Course

University of Antioquia

Practice # 3 - Matrix multiplication using processes

Objectives

- Understand process creation using `fork()`.
- Implement parallel computation using multiple processes and OS APIs.
- Learn shared memory or Inter-Process Communication(IPC) mechanisms (e.g., pipes, shared memory).
- Analyze performance gains from parallelization.

Problem statement

Write a program in **C/C++** and another in **Go** that multiply two large matrices A (size $N \times M$) and B (size $M \times P$) to produce C (size $N \times P$) with the following approaches:

- Sequential: Single-process multiplication (baseline).
- Parallel: Divide work among child processes.

Example:

Matrix A (2×3) Matrix B (3×2) Matrix C (2×2)

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 7 & 8 \\ \hline 9 & 10 \\ \hline 11 & 12 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 58 & 64 \\ \hline 139 & 154 \\ \hline \end{array}$$

Inputs:

- The program should accept matrixes A and B from independent text files.
- The program should accept the number of processes to be implemented to achieve the matrix multiplication.

Output:

- The program print the result matrix C in a text file (C.txt).

Lab steps

Step 1: Sequential implementation

The sequential version implements the matrix multiplication in a single process.

Measure execution time using `time.h` (C/C++) or `time` (Go).

Step 2: Parallel implementation

Process creation:

- Use `fork()` to spawn child processes.
- Each child computes a subset of rows of Matrix C.

Workload distribution:

- Divide N rows of A among K processes.
- Example: For N=100 and K=4, assign 25 rows per process.

IPC mechanism (choose one):

- Shared Memory: Use `shmget()/shmat()` (C/C++) or `shm.Create/shm.Open` (Go) to share matrices.
- Pipes: Send partial results via `pipe()` (C/C++) or `io.Pipe` (Go).

Aggregation:

- Combine results from child processes into C matrix final result.

Step 3: Performance comparison

- Compare execution times of sequential (1 process) vs. concurrent approaches (multi-process).
- Plot speedup vs. number of processes.

Code snippets

Shared memory setup (parent process)

```
#include <sys/shm.h>
// Allocate shared memory for matrices
int shmid = shmget(IPC_PRIVATE, sizeof(int)*N*P, IPC_CREAT | 0666);
int *C = (int*)shmat(shmid, NULL, 0);
```

Child process logic

```
for (int i = start_row; i < end_row; i++) {
    for (int j = 0; j < P; j++) {
        C[i*P + j] = 0;
        for (int k = 0; k < M; k++)
            C[i*P + j] += A[i*M + k] * B[k*P + j];
    }
}
```

Expected output

```
$ ./parallel_matrix_multiply
Sequential time: 12.8 seconds
Parallel time (4 processes): 3.9 seconds
Speedup: 3.28x
```

Deliverables

Code:

- `matrix_mul.c` and `matrix_mul.go`, including in the same code sequential and concurrent implementations (with IPC).

Report:

- Explanation of your IPC choice.
- Performance analysis (table/graph).

Bonus Challenges

- Dynamic Load Balancing: Assign rows to processes on-demand.
- Error Handling: Handle process crashes gracefully.
- Non-Square Matrices: Generalize for any $N \times M \times M \times P$.

References

- OSTEP: Processes API Chapter.
- Linux man pages: `fork()`, `shmget()`, `pipe()`.

How to Submit

- Compress your files into `lab3_processes_<ID>.zip` and upload to the course portal. The ID must belong to any team member.
- Add into the code as comment the team members, including full name and ID.
- Deadline: DD/MM/YYYY.
- **Note:** This lab assumes a Unix-like OS (Linux/macOS). Windows users may use WSL or adapt to WinAPI.

Let me know if you'd like to expand any section (e.g., detailed IPC code, grading rubric)!