Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS3390 (Operating Systems)

First Programming Task

Process & Thread Management

Prepared by:

Name: Ali Shaikh Qasem ID: 1212171

Instructor: Dr. Bashar Tahayna.

Section: 3.

first-semester

2023/2024

Abstract

This task aims to practice multiple processes and threads programming, and apply these concepts to parallelize the matrix multiplication operation. Additionally, to manage the synchronization between threads/processes. Furthermore, to compare the performance of different solutions.

Table of Contents

Naive solution

- In this part, we perform the matrix multiplication using the normal solution without using threads or processes.
- The time complexity of the algorithm is $O(n^3)$.
- We measured the execution time using clock_gettime () function.
- We noticed that the execution time differs in each run so we took about five reads and then we calculated their average. We also applied this technique in all solutions.

The results:

- Execution time = 7.65 (milliseconds).
- Throughput = 130.7 operations per second.

Notice that the execution time is high, so that the naïve solution will not be a good choice.

Multiple processes solution

- In this part, we created multiple processes using fork () function to perform the matrix multiplication.
- We divided the number of rows into the processes such that each process performs the multiplication with same portion size to achieve maximum efficiency for the parallelism.
- For the IPC between processes, we created a shared memory represents the result array, so that each process writes it's result to the shared memory. When they finish their work, the main process prints the whole result array from the shared memory.

We considered different number of processes and the results is shown in the table bellow

Number of processes	Execution time (milliseconds)	Throughput (operations per second)
2	5.03	198.8
4	4.64	215.5
6	6.94	144.1
8	5.05	198.1
10	5.11	195.6

Table 1: processes solution results

- We conclude that all cases give better performance than the naïve solution, this is because we parallelize the operation. The best number of processes is 4 since it gives the least execution time and the best throughput. We also notice that more processes don't always give better performance this is because processes need much resources and memory access which would affect the execution time.

Multiple threads solution

- In this part, we used the pthread POSIX library to create a multi-threaded solution for matrix multiplication.
- We tried different join/detach configurations.

Joinable threads

- In this case we created a pure joinable-threads solution, so that we joined all of them to the main thread in a loop after they were created as shown in the attached code.

We considered different number of threads and the results is shown in the table bellow

Number of threads	Execution time (milliseconds)	Throughput (operations per second)
2	3.84	260.4
4	3.81	262.4
6	3.74	267.4
8	3.45	289.8
10	4.29	233.1

Table 2: joinable threads solution results

- From the results above we notice that all cases give a better performance than the naive solution and the processes solution too, it's because threads don't need resources as much as processes which affects the performance positively. The best number of threads is 8 as it gives the best performance. We also notice that the execution time is going down as the number of threads increases, but when we reach 10 threads it starts rising, after that the more threads applied the less performance is noticed.

Detached threads

- In this case, we created a pure detached-threads solution, so that we didn't have to join them to the main thread, but after creating them we put a loop to wait for them to finish their work, as shown in the code, this is to guarantee that all threads have finished their work so that we get the correct result matrix without errors.

We considered different number of threads and the results is shown in the table bellow

Number of threads	Execution time (milliseconds)	Throughput (operations per second)
2	9.74	102.7
4	5.21	191.9
6	4.55	219.8
8	4.70	212.8
10	5.88	170.1

Table 3: Detached threads solution results

- From the results above, we notice that the performance is reduced a bit compared with the joinable threads, this is probably because waiting each single thread to finish his work

needs more time than joining all threads to the main thread. We also notice that the execution time decreases until we reach a total of 8 threads, after that it starts to increase. The best number of detached threads is 6 which gives the best performance.

Mix of joinable and detached threads

- In this solution, we created a mix of joinable and detached thread to achieve the matrix multiplication, such that half of the threads are detached and the other half is joinable.
- As noticed in the code, we joined the joinable threads to the main thread, while we had to wait for the detached threads until they finished their execution.

Number of threads	Execution time (milliseconds)	Throughput (operations per second)
2	4.84	206.6
4	4.62	216.4
6	4.06	246.3
8	4.65	215.1
10	4.82	207.5

Table 4: Mixed threads solution results

- In this part, we notice that the results are better than the pure detached threads since we have joinable threads too. However, the performance increases until we have 6 threads and after that it begins to decrease. We also conclude that the best performance is achieved with 6 threads.

Results comparison and conclusion

In this task, we learned the concepts of multi-processing and multi-threading by implementing a different solutions of matrix multiplication operation. We have tried different configurations using different number of processes/threads to find an optimal solution for the problem. By analyzing the results mentioned above we can conclude that:

- The best number of child processes is **4** with an execution time = **4.64 (milliseconds).**
- The best number of joinable threads is **8** with an execution time = **3.45 (milliseconds).**
- The best number of detached threads is **6** with an execution time = **4.55 (milliseconds).**
- The best number of mixed threads is **6** with an execution time = **4.06 (milliseconds).**

It's clear from these results that the best solution in general for the problem is the joinable threads solution with 8 threads as it gives the best performance. But we have to take into account that these results might not be very accurate theoretically since we have used a virtual machine to simulate Linux operating system which doesn't achieve high utilization of system resources as it shares the resources with the host machine. Furthermore, these results may differ from device to another because it depends on the specifications of the hardware system like clock speed, number of cores, CPU cache and memory speed.