Birzeit University
Faculty of Engineering and Technology
Department of Electrical and Computer Engineering

OPERATING SYSTEMS ENCS3390
Task-One

Prepared by
Name: Hasan Qarmash
ID: 1210611

Supervised by
Dr.Abdel Salam Sayyad

Birziet-2023

Abstract

The aim of this task to be familiar with process and thread management using POSIX and Pthreads. we will implement process-based and threaded solutions, compare their performance, and analyze trade-offs.

Contents

List of Figures

1 project course

In this task question,involves matrix multiplication. You must create your own performance comparison matrices. Create a 100X100 integer matrix made up of your student number repeated till the matrix is full.Generate another 100X100 integer matrix matrix of (your student number * your birth year) and continue until the matrix is full.Contrast four approaches. In each scenario, time how long it takes to complete the program.

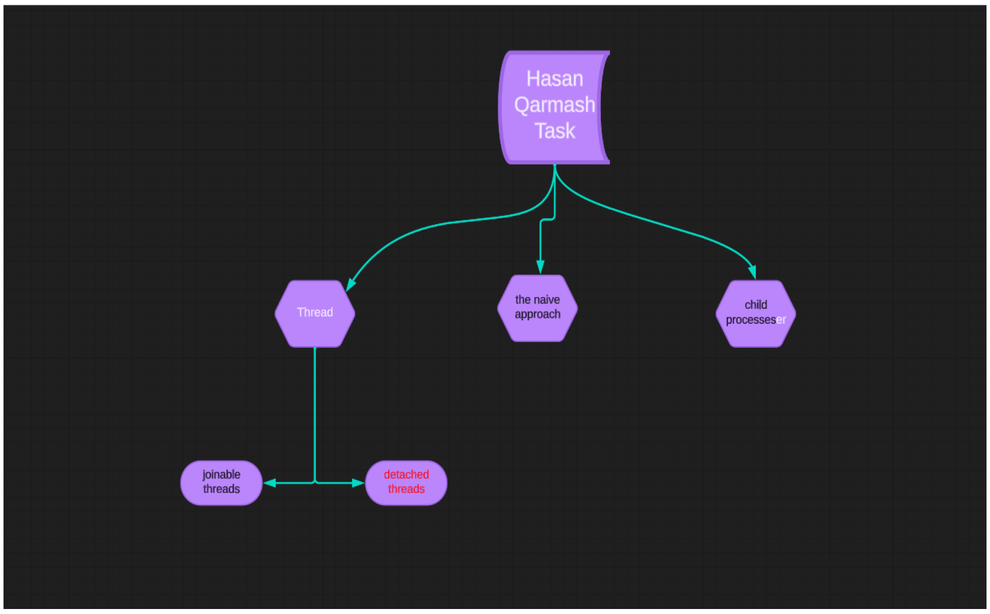


Figure 1: Plan project

- **The naive approach, a program that does not use any child processes or threads.**
- **A program that uses multiple child processes running in parallel.**
- **A program that uses multiple joinable threads running in parallel.**
- **A program that uses multiple detached threads running in parallel.**

2 solution using loop

Hint: all the code in appendix

The first solution using nested loop

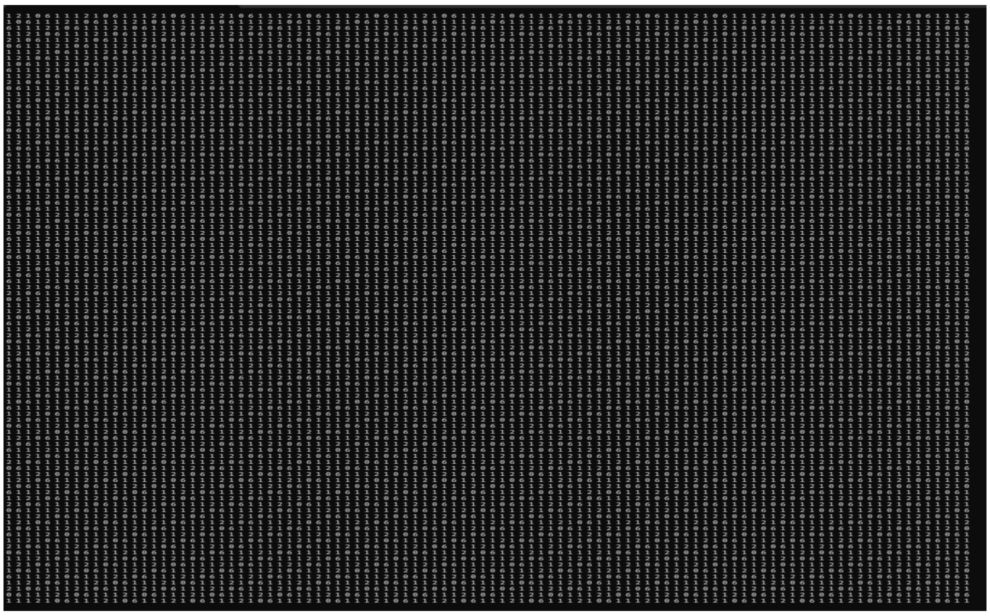


Figure 2: Matrix ID

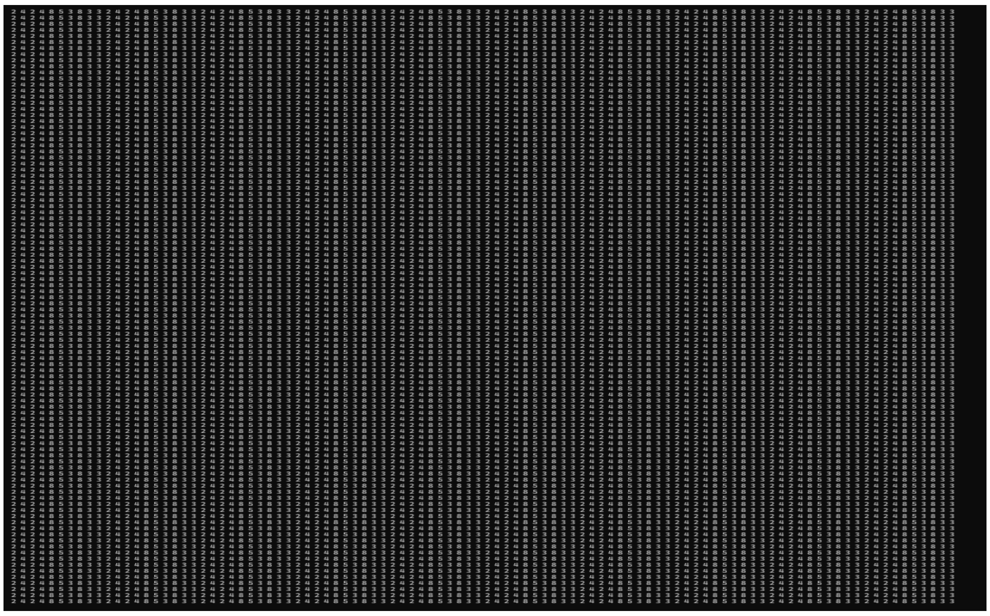
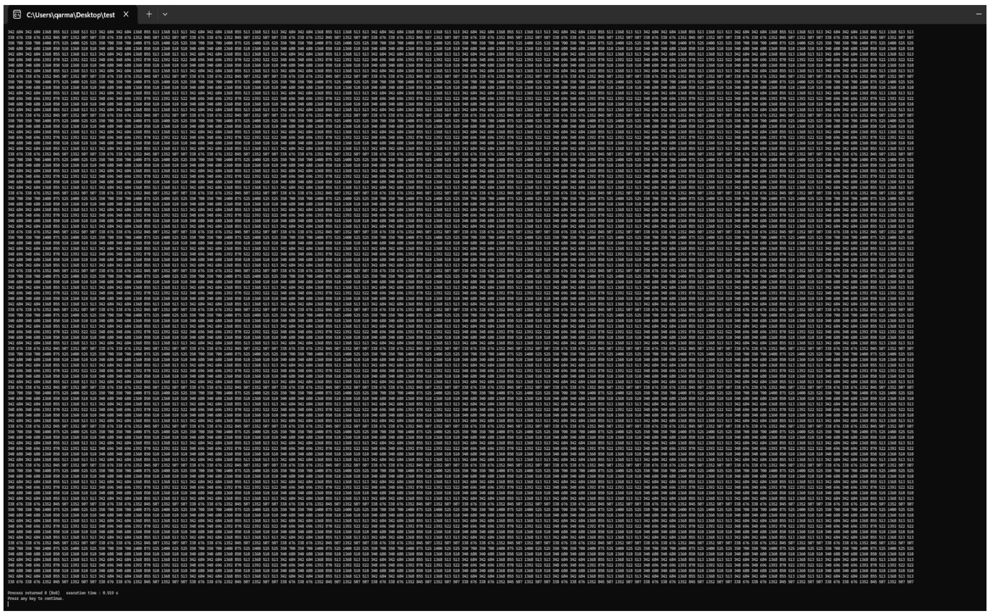
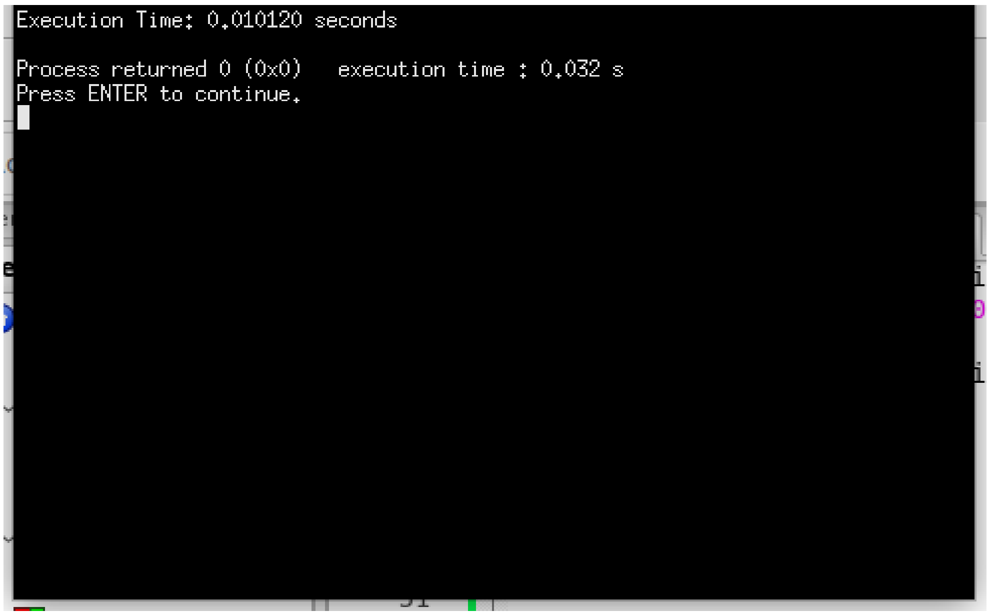


Figure 3: Matrix Year



```
C:\Users\qarma\Desktop\test

Process returned 0 (0x0)   execution time : 0.919 s
Press any key to continue.
```

Figure 4: Result Matrix



```
Execution Time: 0.010120 seconds

Process returned 0 (0x0)   execution time : 0.032 s
Press ENTER to continue.
```

Figure 5: Execution Time first solution

3 solution using Thread

thread is the smallest unit of execution within a process. A process can have multiple threads, and these threads can run concurrently, allowing for parallel execution of tasks. Threads within a process share the same resources (such as memory space), but they have their own program counter, register set, and stack.

Joining Threads:

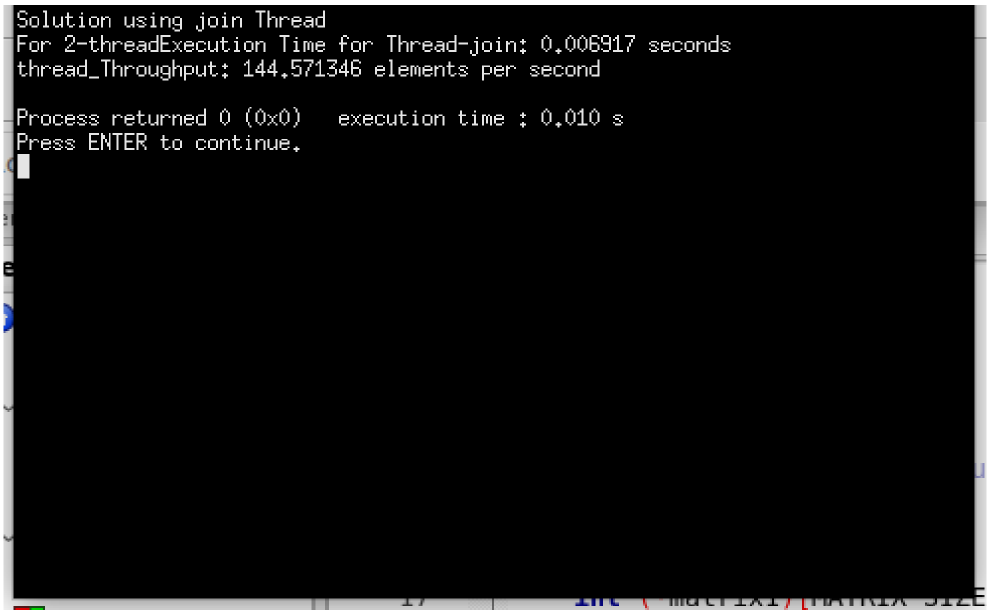
- Purpose: When a thread is created, it often runs concurrently with the main thread or other threads. In some cases, you might want to wait for a particular thread to finish its execution before continuing with the rest of the program.

- Joining Mechanism: The `join` operation is used to accomplish this. When a thread is joined, the calling thread will wait for the specified thread to finish before proceeding. This is useful for scenarios where you need the results or side effects of a thread's execution before moving forward.

Detaching Threads:

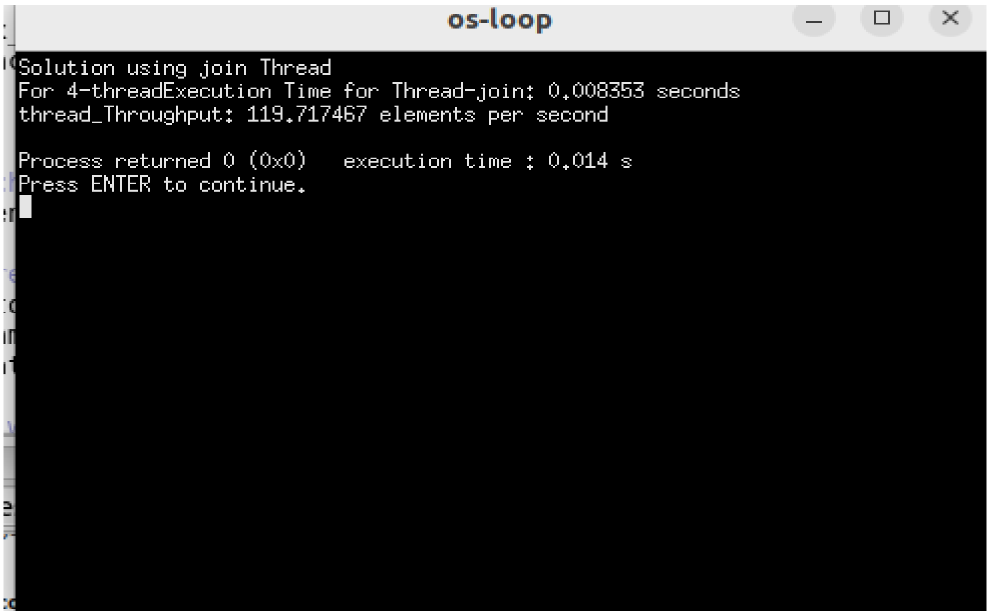
- Purpose: In some situations, you might not be interested in waiting for a thread to finish. For example, if a thread is handling some background task that doesn't affect the main program flow, you might want to let it run independently.
- Detaching Mechanism: The `detach` operation is used for this purpose. Once a thread is detached, its resources are released automatically when the thread exits. You cannot join a detached thread, and it becomes a "daemon" thread that runs in the background.

3.1 using Joining Threads:



```
Solution using join Thread
For 2-threadExecution Time for Thread-join: 0.006917 seconds
thread_Throughput: 144.571346 elements per second

Process returned 0 (0x0)   execution time : 0.010 s
Press ENTER to continue.
```

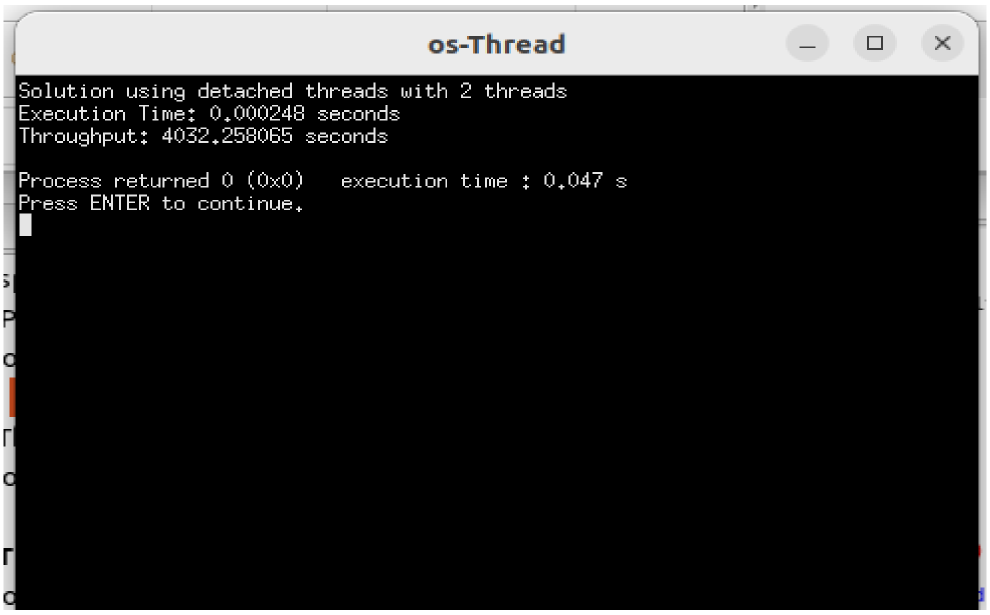
Figure 6: Solution using 2 Joining Threads



```
Solution using join Thread
For 4-threadExecution Time for Thread-join: 0.008353 seconds
thread_Throughput: 119.717467 elements per second

Process returned 0 (0x0)   execution time : 0.014 s
Press ENTER to continue.
```

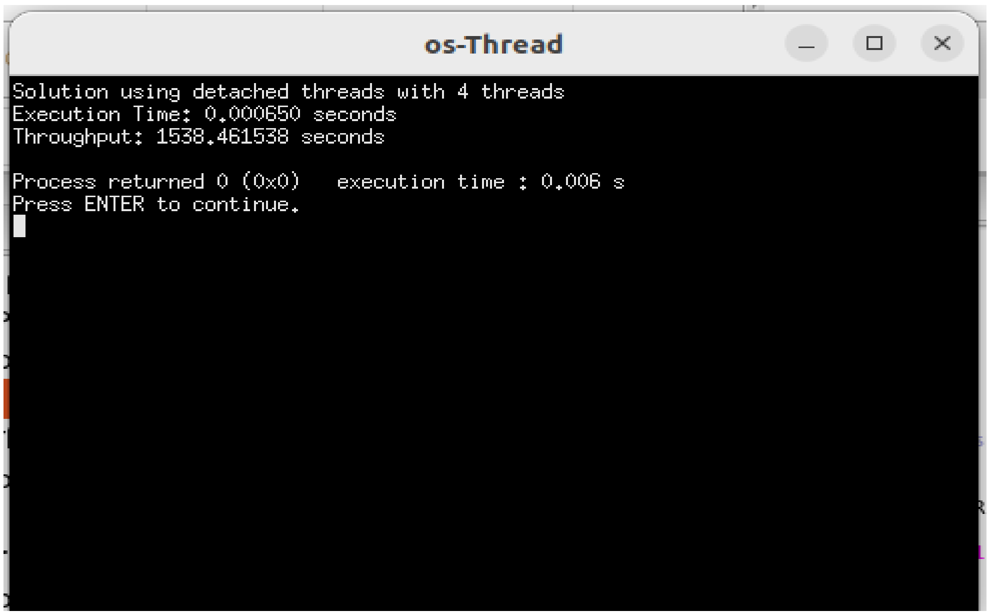
Figure 7: Solution using 4 Joining Threads

3.2 using Detaching Threads:



```
Solution using detached threads with 2 threads
Execution Time: 0.000248 seconds
Throughput: 4032.258065 seconds

Process returned 0 (0x0)   execution time : 0.047 s
Press ENTER to continue.
```

Figure 8: using two Detaching Threads



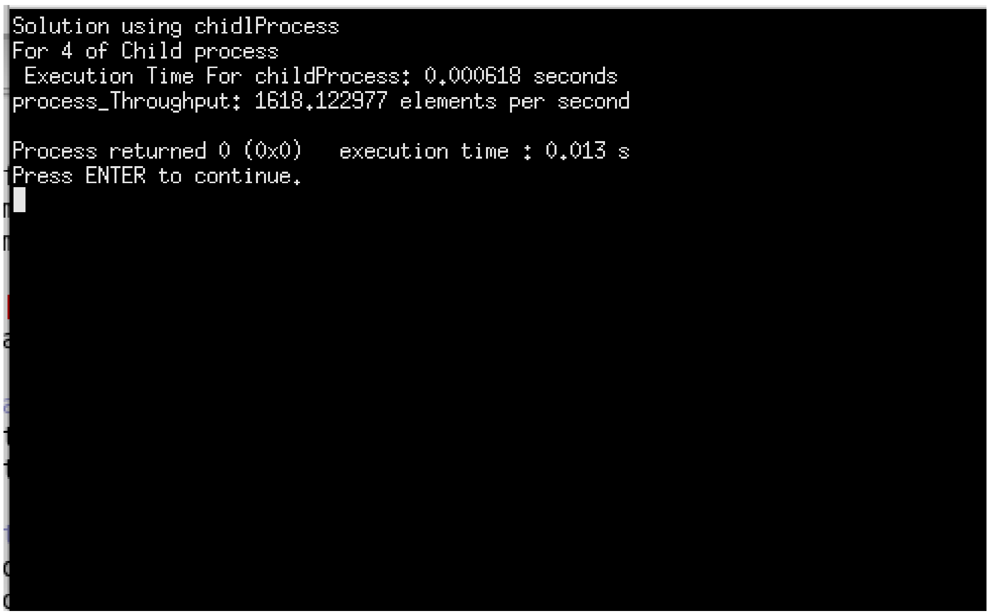
```
Solution using detached threads with 4 threads
Execution Time: 0.000650 seconds
Throughput: 1538.461538 seconds

Process returned 0 (0x0)   execution time : 0.006 s
Press ENTER to continue.
```

Figure 9: using Four Detaching Threads

4 solution using child Process

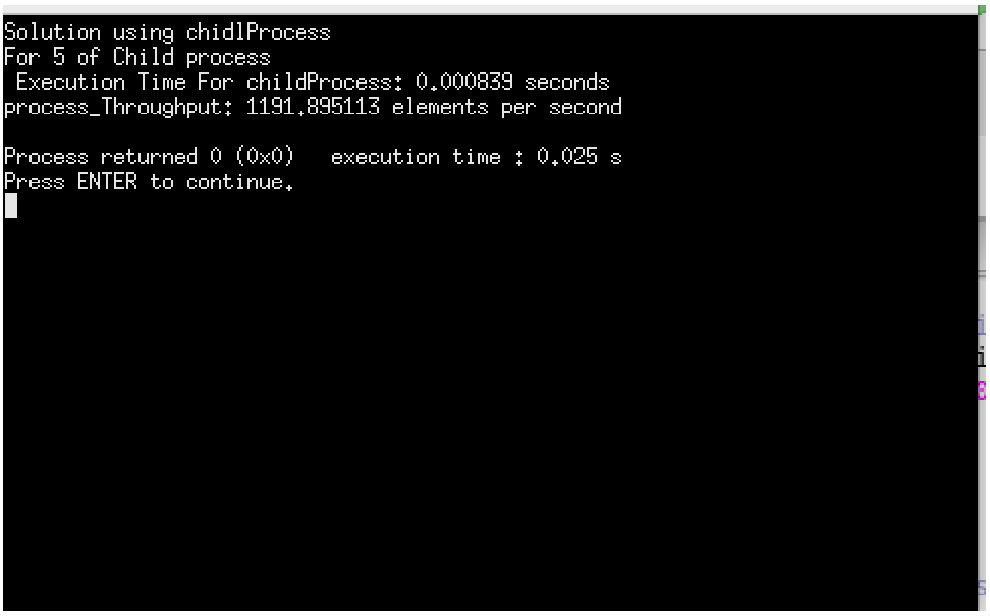
child process is a process that is created by another process, known as the parent process. The parent process typically spawns the child process to perform a specific task, and both processes may run concurrently. The child process is a separate execution unit with its own memory space, resources, and execution context.

- Communication between a parent and child process can be established through Inter-Process Communication (IPC) mechanisms, such as pipes, shared memory, or message queues.
- IPC allows data exchange between the parent and child processes, enabling coordination and collaboration.
- Child processes can terminate independently of the parent process. They may exit when their task is complete or encounter an error.
- The termination of a child process does not necessarily affect the parent process, unless the parent is actively monitoring the child's status.



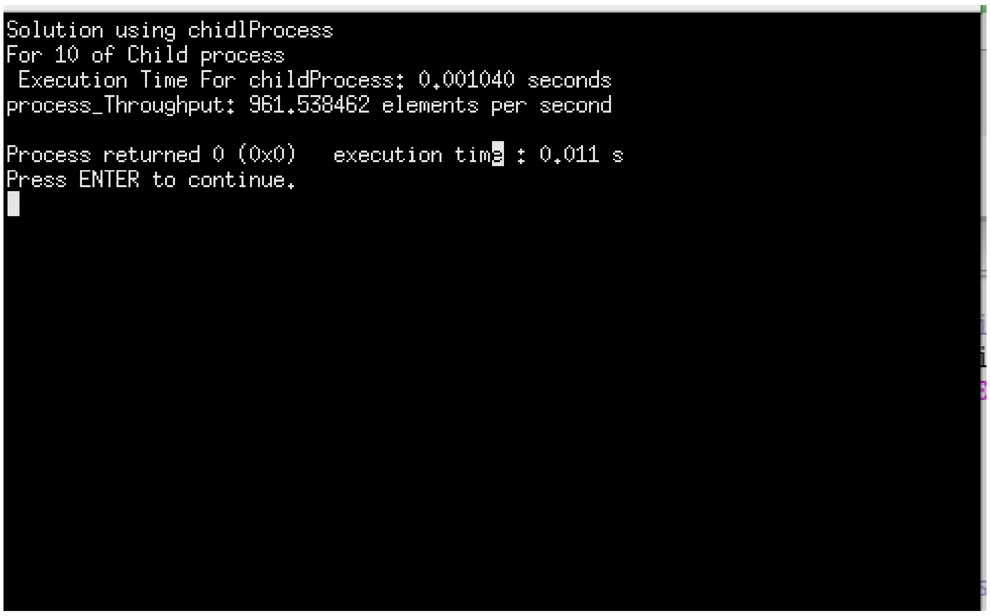
```
Solution using chidlProcess
For 4 of Child process
 Execution Time For childProcess: 0.000618 seconds
process_Throughput: 1618.122977 elements per second

Process returned 0 (0x0)   execution time : 0.013 s
Press ENTER to continue.
```

Figure 10: Execution time and Throughput with 4 child process



```
Solution using chidlProcess
For 5 of Child process
 Execution Time For childProcess: 0.000839 seconds
process_Throughput: 1191.895113 elements per second

Process returned 0 (0x0)   execution time : 0.025 s
Press ENTER to continue.
```

Figure 11: Execution time and Throughput with 5 child process



```
Solution using chidlProcess
For 10 of Child process
 Execution Time For childProcess: 0.001040 seconds
process_Throughput: 961.538462 elements per second

Process returned 0 (0x0)   execution time : 0.011 s
Press ENTER to continue.
```

Figure 12: Execution time and Throughput with 10 child process

Hint: When we increase the number of child process, the execution time increases and Throughput decreases

5 Explanations results & analysis

In this section we will learn more thing ,First, we made three nested loop , and the execution time was =0.01 sec, knowing that (Complexity Time) = O(n*n*n), then we made Thread with its two types **join** and **detach**, and the results were as follows:
For join Thread

- Join Thread execution time with 4 threads =0.008353 sec
- Join Thread execution time with 2 threads =0.000248 sec
- Join Thread Throughput with 4 threads =119.7 element per second
- Join Thread Throughput with 2 threads =144.57 element per second

For Detaching Threads

- Detaching Threads execution time with 2 threads =0.000248 sec
- Detaching Threads execution time with 4 threads =0.000650 sec
- Detaching Threads Throughput with 2 threads =4032.2 element per second
- Detaching Threads Throughput with 4 threads =1538.46 element per second

and when We did the last solution, which is **child Process**. The results were

- child Process execution time with 4 child =0.000618 sec
- child Process execution time with 5 child =0.000839 sec
- child Process execution time with 10 child =0.00104 sec
- child Process Throughput with 4 child =1618.122 element per second
- child Process Throughput with 5 child =1191.8 element per second
- child Process Throughput with 10 child =961.538 element per second

for all this result we learn Threads with join have better performance with 2 threads than with 4 threads. Detached threads have higher throughput with 2 threads compared to 4 threads. Child processes show a decrease in throughput as the number of child processes increases, which is expected due to the overhead of managing multiple processes.
the choice between threads and processes depends on the nature of the task, the level of parallelism achievable, and the overhead associated with managing threads or processes.but In this task i prefer Thread detaching over the rest of the methods because the function of multiplying two matrices is an operation that has the same shared data but each threas works individually and this is what makes it take the advantage of process in unity and take the advantage of join threads in shared data

6 Conclusion

In this project, we Learning Objectives Hands-on practice with processes, threads, synchronization, performance comparison.We faced problems downloading the program on Linux and how to work on it, as I had never used it before, but I overcame all of these problems, and I had no knowledge of how to use the topics I had learned through programming, so I watched several lectures for that.
With Best Wishes:) .