**Operating Systems**

**Project Report**

**Course Code: CS2006**

**Project Name:**

**Comparison between**

**MultiProcessing and MultiThreading**

Instructor: Dr. Nausheen Shoaib

Group Members:

K21-3272 Huzaifa Quaid

K21-3333 Saleh Shamoon

K21-3232 Ahmed Saif

BCS 4C

**OBJECTIVE:**

Our project mainly focuses on the differences between multithreading and multiprocessing. The performance comparison is observed by the implementation of 3 algorithms/ situations i.e.

1. Counting Number of Primes
2. Matrix Multiplication
3. Travelling Salesman Problem

All 3 of these problems are tackled in 2 different ways one being via multiprocessing and the other being multithreading. The implementation of the above mentioned problems is almost the same for both the cases but with a slight difference of syntax.

**PLATFORM AND LANGUAGE:**

● The Operating System used is Ubuntu (Linux).

● The languages used for the project are a mixture of C & C++.

**INTRODUCTION & APPLICATIONS:**

**MultiProcessing:**

Multiprocessing refers to the use of multiple processors or cores to execute tasks simultaneously. In a multiprocessing system, a program can be divided into multiple processes, each of which can be executed on a different processor or core. This allows the program to perform multiple tasks in parallel, improving overall performance.

In general, multiprocessing is better suited for applications that can be easily parallelized, such as scientific simulations and rendering tasks.

Some examples of where multiprocessing is/ can be applicable:

1. Video encoding: Video encoding is a computationally intensive task that can benefit greatly from multiprocessing. Video encoding software can divide the task of encoding a video file into multiple processes, with each process working on a different section of the video.

1. Computational simulations: Many scientific simulations involve complex calculations that can be divided into smaller sub-tasks that can be executed in parallel. Multiprocessing can be used to distribute these sub-tasks across multiple processors, reducing the overall simulation time.
2. Image processing: Image processing software can use multiprocessing to speed up tasks such as image filtering, image enhancement, and object recognition. Multiple processes can be used to apply filters or perform other operations on different parts of an image simultaneously.
3. Machine learning: Many machine learning algorithms, such as neural networks, involve processing large amounts of data. Multiprocessing can be used to distribute the processing of this data across multiple processors, speeding up the training process.
4. Data analysis: Data analysis software can use multiprocessing to speed up tasks such as sorting, filtering, and aggregating large datasets. Multiple processes can be used to work on different parts of the data simultaneously, reducing the overall processing time.

**MultiThreading:**

Multithreading refers to the use of multiple threads within a single process to perform different tasks simultaneously. Each thread has its own program counter, stack, and set of registers, allowing it to execute independently of other threads within the same process. This allows a program to perform multiple tasks in parallel without the overhead of creating multiple processes.

In general, multithreading is better suited for applications that involve many small tasks that can be executed concurrently, such as web servers and database applications.

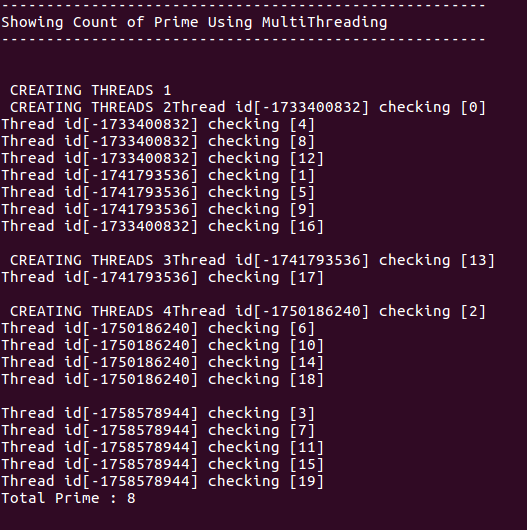
Some examples of where multithreading is/ can be applicable:

1. Web servers: A web server can use multithreading to handle multiple requests from clients simultaneously. Each request can be handled by a separate thread, allowing the server to process multiple requests concurrently.
2. GUI applications: Graphical user interface (GUI) applications often use multithreading to keep the user interface responsive while performing long-running tasks in the background. For example, a video editing application may use a separate thread to encode a video while still allowing the user to interact with the application.
3. Game engines: Game engines often use multithreading to handle tasks such as physics calculations, AI, and rendering. Each task can be handled by a separate thread, allowing the game engine to perform multiple tasks concurrently and maintain a smooth gameplay experience.
4. Database applications: Database applications can use multithreading to handle multiple queries from clients simultaneously. Each query can be handled by a separate thread, allowing the application to process multiple queries concurrently.
5. Media players: Media players can use multithreading to handle tasks such as decoding audio and video files while still allowing the user to interact with the player. For example, a media player may use a separate thread to decode an audio file while still allowing the user to control playback.

**METHODOLOGY:**

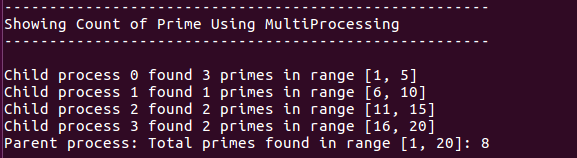
* Creating 3 different programs for the aforementioned 3 different problems and implementing multiprocessing and multithreading for all the problems.
* Having the same number of processes and threads created for each problem so as to make comparison easy and understandable.
* Collecting time data for all 6 parts (3 problems \* 2 methods) and evaluating the trade-offs between Multiprocessing and Multithreading.

**Counting Number of Primes:**



In this program, 4 threads are being created which are then distributing work among each other and each thread is checking whether every 4th number from its starting position is a prime or not. The range in this image is set to 20 however, the program works on any given range.

The id’s being assigned to the thread are being automatically given by the operating system.



The same functionality but using processes. Here it can be observed how the load is being divided among 4 different processes and the number of primes found is being displayed by the parent process.

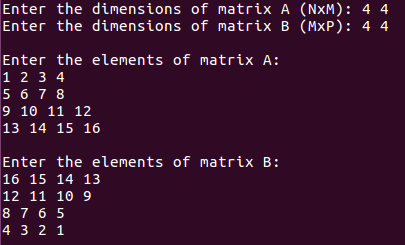
The number of primes found in both is 8, which is correct as evident by

[*“What are the prime numbers? There are 8 prime numbers under 20: 2, 3, 5, 7, 11, 13, 17 and 19.”*](https://thirdspacelearning.com/blog/what-is-a-prime-number/#:~:text=What%20are%20the,17%20and%2019.)



The image attached above represents the time it took for both methods to complete this task. Multiprocessing proved to be faster in this scenario since the workload was equally distributed among 4 processes and there was no need to wait for one process to finish for another to start as there was a need in multithreading. Since the processes were working simultaneously, it took lesser time to complete the task.

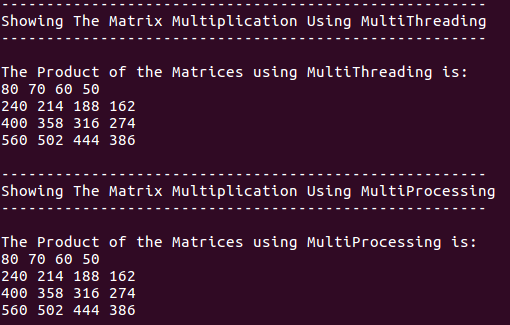
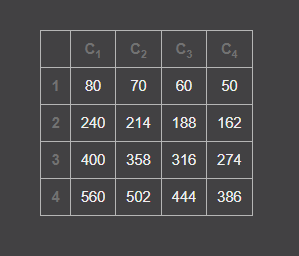
**Matrix Multiplication:**



In Matrix Multiplication, the same thing as before is taking place i.e. distribution of work among 4 threads and 4 processes with the difference being the use of shared memory (to make Inter Process Communication possible in order to display correct output).

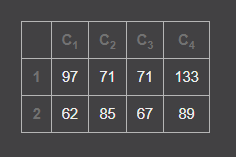
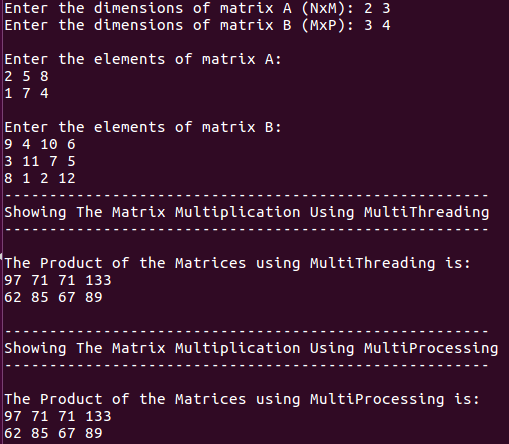
In the example provided above, the situation for thread and processes is ideal as the work is being equally distributed among all 4 threads and 4 processes. Each thread and each process is multiplying 1 row of matrix A with 1 column of matrix B and is then terminating.

The matrices provided by the user are being used in both methods instead of having to typing input material twice.



The answer was verified using an online [matrix calculator](https://matrix.reshish.com/multCalculation.php).

The methods work perfectly fine in case of dissimilar dimensioned matrices as well.

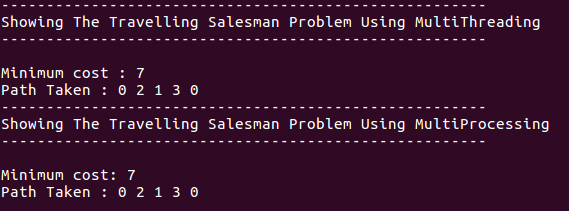


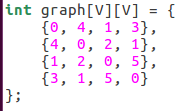
This proves that both the methods work correctly.



We can observe that in both iterations of this program, the time it took for Multithreading to complete the task was much longer than the time MultiProcessing took to complete it. The reason for this is the same as the task before i.e. multiprocessing assigns different rows of the input matrix to different processes, whereas multithreading assigns different portions of the input data to different threads. In multiprocessing, load balancing is better and contention issues (deadlocks) are avoided, while in multithreading, there is a higher chance of cache misses and the overhead of thread creation and synchronization can affect performance.

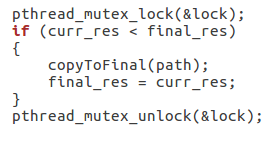
**Travelling Salesman Problem:**





In the Travelling Salesman Problem, an approach similar to matrix multiplication was used when it came to Inter Process Communication so as to store the desired path in a dynamically created array. However, the difference in approach here cam in with multithreading, where we used mutex (a semaphore) in order to restrict access in a specified area which we call the critical zone. The critical zone is the portion of code where we store the travelling cost from each starting point of the salesman to every other city.

The Critical Zone:



The image of the output provided above clearly show that the minimum cost of travel from city 0 to 3 (or 1 to 4) is “7” with the best possible path being:

0 🡪 2 🡪 1 🡪 3 🡪 0

The path taken visits every city exactly once and brings the salesman back to the point of origin. Thus, solving the Travelling salesman problem.



The reason for MultiProcessing being faster than MultiThreading in this situation as well reamins the same. The multiple created processes are able to work simultaneously in multiple cores, making the execution faster. While the threads, even though working in parallel, are slower because they are active in a single core and have to be on hold/ wait because of contention/ deadlock issues.

**Situations where:**

**MultiThreading > MultiProcessing?**

Multithreading is more suitable for I/O-bound applications, where input/output operations are the bottleneck. It allows for overlapping I/O operations and prevents the program from blocking. Examples of such applications are web servers and GUIs. Multithreading is effective for executing many independent tasks in parallel.

**MultiProcessing > MultiThreading?**

Multiprocessing is generally better than multithreading in scenarios where the application is CPU-bound, rather than I/O-bound. This means that the tasks require significant CPU processing. Using multiple processes can help increase overall efficiency, as the processes can be executed on separate CPUs or cores and can work on different tasks in parallel. For example, scientific computing applications that perform complex calculations on large datasets can benefit from multiprocessing.

**Conclusion:**

In the algorithms/ problems we tried to solve and provide a difference between Multiprocessing and Multithreading with, it was proven that Multiprocessing was the better approach in these situations. The reason behind this was CPU optimization which gives Processes a great advantage over Threads. However, as mentioned above there are certain situations where Threads would prove to be superior to Processes. Although there also can be (and are) situations where both MultiProcessing and MultiThreading work together to achieve better results.

Some examples are:

1. Image processing: When performing image processing tasks such as resizing, filtering, or enhancing images, a combination of multiprocessing and multithreading can be used. Multiprocessing can be used to split the workload into multiple processes, while multithreading can be used within each process to further parallelize the computation.
2. Video encoding: When encoding a video, multiprocessing can be used to split the video into chunks and encode each chunk in parallel, while multithreading can be used within each process to speed up the encoding process even further.
3. Scientific simulations: Scientific simulations often involve complex computations that can benefit from both multiprocessing and multithreading. Multiprocessing can be used to split the simulation into multiple processes, while multithreading can be used within each process to speed up the computations.
4. Web server: When handling multiple requests from clients in a web server, multiprocessing can be used to handle each request in a separate process, while multithreading can be used within each process to handle multiple requests concurrently.
5. Machine learning: When training a machine learning model, multiprocessing can be used to split the training data into multiple processes, while multithreading can be used within each process to speed up the gradient computations. This can result in faster training times and better model performance.

The End