Faculty of Engineering and Technology
Electrical and Computer Engineering Department

Operating Systems
ENCS33990

Task1
Process and Thread Management

Prepared by:
Abdel Rahman Shahen 1211753

Partners:
no partners

Instructor: Dr. Abdel Salam Sayyad

Section: 2
Date: 11/24/2023

Theory:

The provided C code implements matrix multiplication using three different approaches: a naive sequential method, a parallelized version using pthreads (POSIX threads), and a multiprocess approach using fork() and shared memory. The matrices are generated using predefined content arrays, and the results are printed along with the execution time for each method.

The naive sequential approach calculates matrix multiplication straightforwardly with nested loops. The pthreads implementation divides the task among multiple threads for parallel execution, enhancing performance. The multiprocess approach relies on fork() to create child processes that independently compute submatrices in parallel, and shared memory facilitates communication between processes.

These approaches demonstrate the trade-offs between simplicity and performance optimization in matrix multiplication, offering insights into the benefits of parallelization and multiprocessing in computational tasks.

In this experimental context, the optimal configuration for matrix multiplication was found to be 10 processes and 10-15 threads for larger matrices (size: 1000), leveraging the benefits of parallel processing. However, for smaller matrices (size: 100), a more efficient performance was observed with 2-4 threads and 2-3 processes. This nuanced approach takes into account the matrix size, demonstrating that a tailored balance of threads and processes is crucial for optimizing parallelization benefits. The experiment reveals that, while a higher thread count is effective for larger matrices, a more conservative configuration yields better results for smaller matrices, emphasizing the importance of adaptability in parallel computing strategies.

About/Approach	Naïve	Multiprocess	Multithread (joined threads)	Multithread (detached threads)
Time1 (size:1000)	3.91s	1.29s	1.63s	0.16s
Time2 (size:1000)	3.92s	1.14s	1.45s	0.20s
Time3 (size:1000)	3.75s	1.14s	1.52s	0.20s
Time4 (size:1000)	3.79s	1.06s	1.55s	0.18s
Time5 (size:1000)	3.86s	1.16s	1.39s	0.20s
Average time (size:1000)	3.846s	1.158s	1.508s	0.188s
Time1 (size:100)	0.003s	0.005s	0.006s	0.001s (zeros)
Time2 (size:100)	0.003s	0.002s	0.007s	0.002s(zeros)
Time3 (size:100)	0.003s	0.003s	0.006s	0.002s(zeros)
Time4 (size:100)	0.004s	0.003s	0.006s	0.003s
Time5 (size:100)	0.004s	0.002s	0.005s	0.002s
Average time (size:100)	0.0034s	0.004s	0.006s	0.002s

Summary:

In the experiments, the naive approach consistently exhibited the longest execution time, indicating its sequential nature. The multiprocess approach showed significant improvement over the naive method for larger matrices (size: 1000), leveraging parallel processes. The joined threads in multithreading also demonstrated notable efficiency gains, while detached threads exhibited the fastest performance, particularly for smaller matrices (size: 100). Overall, the average times highlight the superiority of the multiprocess and multithreaded (detached threads) approaches in optimizing matrix multiplication tasks, showcasing the benefits of parallelization for enhanced computational efficiency.

Detached threads, while providing efficiency in some parallel computing scenarios, are not ideal for matrix multiplication due to their lack of synchronization control. In matrix multiplication tasks, data dependencies are critical, and detached threads lack the ability to ensure synchronized execution, leading to potential data inconsistencies and errors. Joined threads offer a more controlled and coordinated approach, ensuring accurate computation and maintaining the integrity of the results in complex tasks like matrix multiplication.