OS – Assignment 2

Nofar Selouk (318502721) & Noam Tarshish (207761024)

Q1. Amdahl's Law (5 points)

Amadhal's Law: $speedup \leq \frac{1}{S+\frac{(1-S)}{N}}$

- S – Serial portion of the program
- 1-S – Parallel portion of the program
- N - number of processing cores

1. 1-S = 0.75, S = 0.25, N=4

$$speedup \leq \frac{1}{0.25+\frac{0.75}{4}} \rightarrow \boldsymbol{speedup \leq 2.29}$$

The maximum speedup using 4 processors based on this program is 2.29

2. 1-S = 0.6, S = 0.4, Speedup = 3

$$3 \leq \frac{1}{0.4+\frac{0.6}{N}} \rightarrow 1.2+\frac{1.8}{N} \leq 1 \rightarrow \frac{1.8}{N} \leq -0.2 \rightarrow N = \frac{1.8}{-0.2} = \boldsymbol{-9} \rightarrow \boldsymbol{not\ possible}$$

Since the number of processors cannot be negative, it is concluded that achieving a speedup of 3 with the given serial and parallel portions of the task is not possible.

3. 1-S = 0.9, S = 0.1, N=32

$$speedup \leq \frac{1}{0.1+\frac{0.9}{32}} \rightarrow \boldsymbol{speedup \leq 7.805}$$

The maximum speedup using 32 processors based on this program is 7.805

4. 1-S = 0.85, S = 0.15, N = ∞

$$speedup \leq \frac{1}{0.15+\frac{0.85}{\infty}} \rightarrow speedup \leq \frac{1}{0.15+0} \rightarrow \boldsymbol{speedup = 6.667}$$

The maximum speedup using infinity processors based on this program is 6.667

5. 1-S = 0.8, S = 0.2

for 4 processors: $speedup \leq \frac{1}{0.2+\frac{0.8}{4}} \rightarrow speedup = 2.5$

for 32 processors: $speedup \leq \frac{1}{0.2+\frac{0.8}{32}} \rightarrow speedup = 4.445$

The speedup increased from 2.5 to 4.445 when the number of processors is increased from 4 to 32 and based on this program.

Q2. Process vs. Thread (8 points)

1. Results:

Process Time (ms)	Thread Time (ms)
67.6833	1.2289
50.4948	0.6043
39.0753	0.6022
41.1198	0.6109
37.7906	0.6258
37.94	0.5014
38.7091	0.6055
40.8306	0.7729
39.4848	0.6243
43.9986	0.4451

2. The results demonstrate significant differences in execution times between the two methods, highlighting the performance characteristics of threads and processes. analysis:

Category	Processes	Threads
Execution Time	Higher – values ranging from 37.79ms to 67.6833ms	Lower - values ranging from 0.4551ms to 1.2289ms
Consistency	Variability – large range of values. This is due to the overhead associated with creating and managing a new process.	Consistent - reflecting the lower overhead in creating and managing threads.
Space	Isolated - Each process has its own memory space, which makes inter-process communication (IPC) more complex	Same - Threads within the same process share the same memory space, making communication between threads faster and more efficient.
Overhead	Higher - Creating a new process involves duplicating the current process's memory space, which is a relatively expensive operation.	Lower - Creating a new thread is less expensive than creating a new process. It does not require duplicating the memory space, resulting in lower overhead
Context Switching	Slower - Switching between processes requires saving and loading different memory contexts, which adds to the execution time.	Faster - Switching between threads is faster compared to switching between processes because threads share the same memory context.

Q3. Multithreaded Matrix Multiplication (15 points)

Name	PID	Status	User name	CPU	Memory (ac...	Threads	Architec...	Description
MatrixMultiplier.exe	2712	Running	noamt	38	46,964 K	21	x64	MatrixMultiplier

Q4 .Multithreaded MergeSort (12 points)

a) The multi-threaded merge-sort algorithm leverages the power of parallel processing to sort large datasets more efficiently. The basic idea is to divide the array into smaller subarrays, sort these subarrays concurrently using multiple threads, and then merge the sorted subarrays.

Steps:

1. **Divide**: Split the array into two halves.
2. **Conquer**: Recursively sort each half. If the size of a half is greater than a specified threshold, create a new thread to sort that half; otherwise, sort it in the current thread.
3. **Merge**: Merge the two sorted halves into a single sorted array.

This approach, known as "fork and join," involves "forking" the task into smaller sub-tasks that are processed in parallel and then "joining" the results.

b)