CS342 Operating System

Fall 2023

Project #1

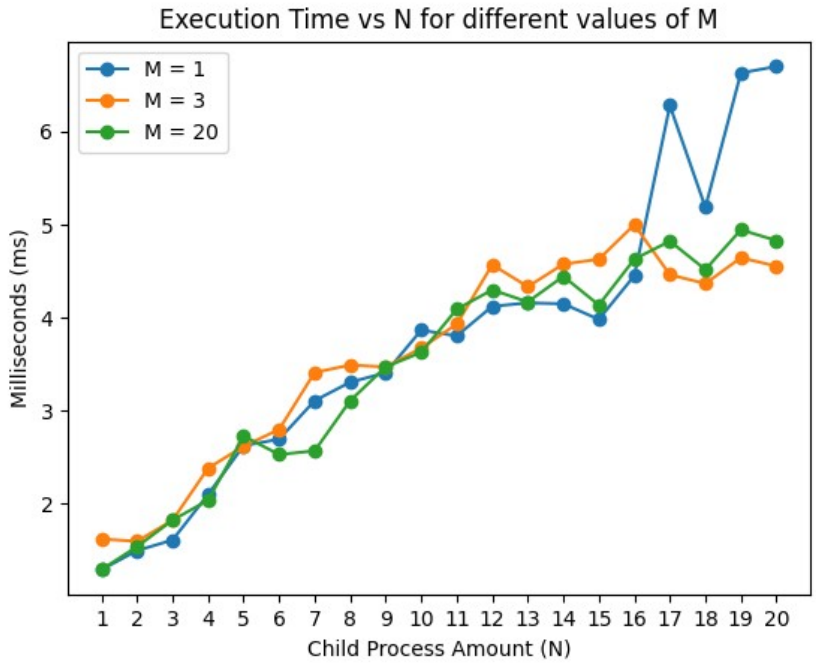
Full Name: Arda İynem

Student ID: 22002717

Date: 20.10.2023

1. Concurrent Processes and Message Queues

1.1 Varying Child Process Amount (N)



	1	3	20
1	1.293	1.623	1.295
2	1.499	1.599	1.541
3	1.612	1.825	1.825
4	2.101	2.384	2.04
5	2.626	2.619	2.728
6	2.695	2.8	2.531
7	3.109	3.412	2.57
8	3.308	3.492	3.109
9	3.409	3.472	3.472
10	3.867	3.675	3.628
11	3.805	3.931	4.092
12	4.123	4.57	4.3
13	4.16	4.334	4.17
14	4.15	4.576	4.445
15	3.983	4.629	4.137
16	4.452	5.0	4.63
17	6.28	4.462	4.823
18	5.186	4.367	4.518
19	6.63	4.645	4.945
20	6.7	4.554	4.829

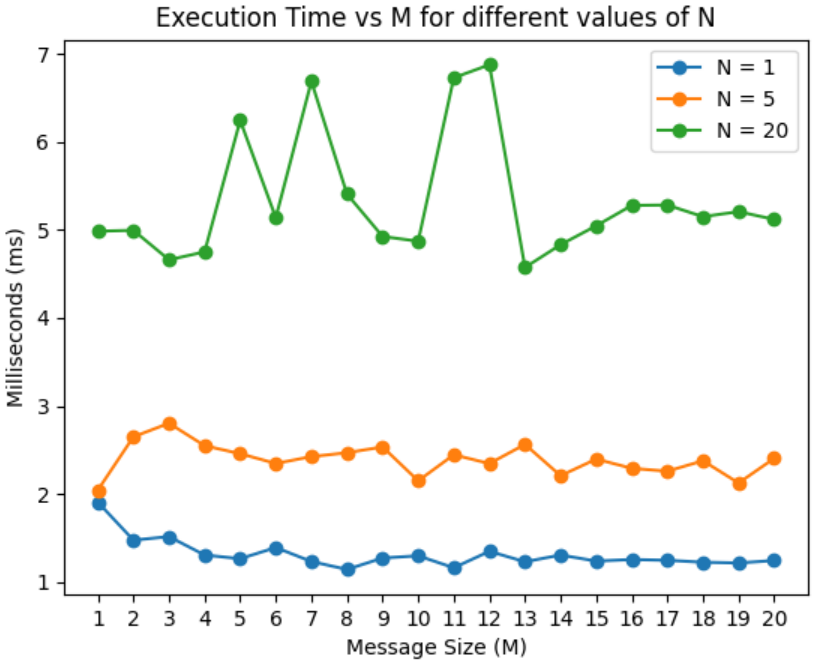
Mean: 3.60975

As the graph shows, when the child process amount is increased, the execution time of the program also increases for all 3 values of M. Execution time plots have a positive trend, but the increase is noisy due to the fact that real execution time is measured, and blocking operations and other intercepting processes' execution time cause variations in the real execution time.

Although the data amount is not enough to draw a certain conclusion, it can be seen that message size generally doesn’t have a great effect on execution time for a given N. The exception is when the message size is very low, and the child process amount is high (the plot line becomes noisy) since the message queue becomes full and blocks message sending until some message is received.

In summary, for this specific program—which is rather simple and does not require multiprogramming due to the limited data—the overhead caused by multiple child processes is not compensated by the benefits that multiprogramming brings.

1.2 Varying Message Size (M)



	1	5	20
1	1.907	2.043	4.985
2	1.475	2.649	4.995
3	1.515	2.803	4.66
4	1.306	2.548	4.75
5	1.264	2.457	6.244
6	1.391	2.346	5.138
7	1.232	2.424	6.688
8	1.142	2.468	5.407
9	1.272	2.534	4.925
10	1.297	2.149	4.871
11	1.161	2.445	6.728
12	1.348	2.345	6.876
13	1.229	2.563	4.573
14	1.306	2.207	4.832
15	1.237	2.396	5.046
16	1.255	2.291	5.278
17	1.246	2.258	5.282
18	1.224	2.377	5.15
19	1.216	2.124	5.207
20	1.243	2.404	5.119

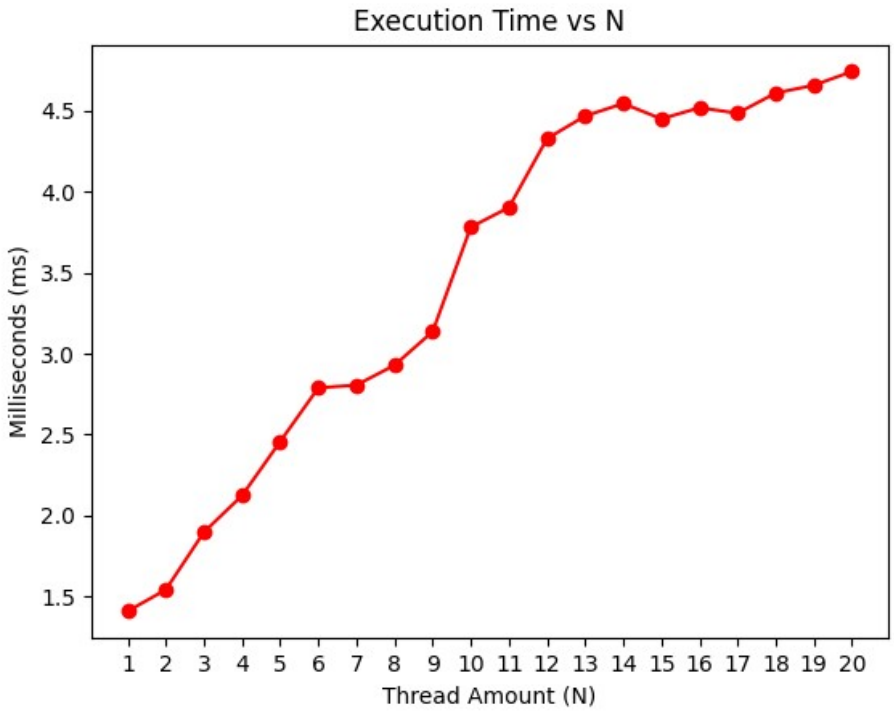
Similar to the previous graph outcomes, varying the independent variable M does not necessarily increase the execution time for all 3 values of N, as each plot line remains relatively flat with some noise. Again, execution time plots are noisy due to the fact that real execution time is measured, and blocking operations and other intercepting processes' execution times cause variations in the real execution time.

However, this graph emphasizes the significance of the uncompensated overhead that child processes cause. It can be clearly seen that as the number of child processes is increased, the execution time of the program also increases due to the fact that this program cannot effectively utilize multiprogramming, as explained in the previous section.

Lastly, it is notable to mention that when the child process amount is high and message size is relatively low, the plot line becomes noisier as the message queue becomes full and blocks message sending until some message is received. This corresponds to the observations made in the previous graph.

2. Threads

2.1 Varying Thread Amount (N)



1	1.407
2	1.54
3	1.896
4	2.122
5	2.456
6	2.788
7	2.805
8	2.928
9	3.136
10	3.782
11	3.903
12	4.329
13	4.469
14	4.546
15	4.451
16	4.52
17	4.486
18	4.611
19	4.659
20	4.744

Mean: 3.4789

As the graph shows, when the thread amount is increased, the execution time of the program also increases. Execution time plots exhibit a positive trend, but the increase is noisy due to the fact that real execution time is measured, and blocking operations and other intercepting processes' execution time cause variations in the real execution time.

For this specific program, which is rather simple and does not require multithreading due to the limited data, the overhead caused by multiple threads is not compensated by the benefits that multithreading brings.

Finally, even though both multiple child processes and multiple threads are not advantageous for this specific program and data, when both multiprogramming methods are compared, **utilizing threads is better in terms of execution time**. This is due to **no context switches (thread switches are very efficient)** and **less overhead of threading approach as program data is shared but not copied**, in contrast to creating processes. This execution time difference between the two methods can be seen by **comparing the mean execution time of both approaches**, which highlights the superiority of the thread approach.