Name : Kakadiya Omikumar A.

Roll no : 24M0789

Report for question 4b:

0.Threads are fun

For program threads.c

```
PID of Foo: 322614
PthreadID : 140443651430080
Value of x: 7
PID of Bar: 322614
PthreadID : 140443643037376
Value of x: 7
```

Same PID: Both threads (Foo and Bar) have the same Process ID (PID: 322614), which shows that threads share the same process space.

Different PthreadIDs: The threads have different Pthread IDs, confirming that they are distinct threads running within the same process

Shared Memory: Both threads see the same value of x (7**)**, demonstrating that threads share the same memory space. Changes to shared variables made by one thread are visible to other threads in the same proces.

For processes.c

```
PID of Process: 333456

Value of x: 2

PID of Process: 333457

Value of x: 7

Parent done
```

Different PIDs: The two processes have different Process IDs **(**which indicates that they are separate processes

Separate Memory Space: The processes have different values for x (7 and 2). This demonstrates that each process has its own separate memory space, and changes in one process do not affect the memory of another process

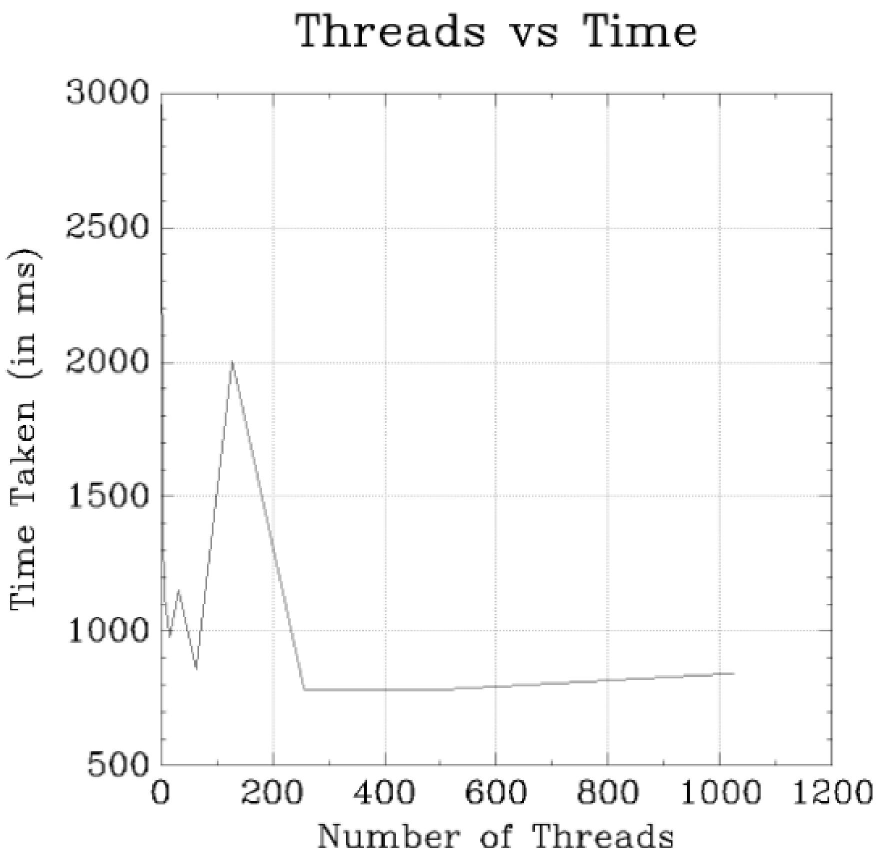
Task1B:TheSweetSpot

When the number of threads is small ,there's an initial increase in time. This can be due to the overhead associated with creating and managing threads. Each new thread requires resources such as stack memory and its own context .As more threads are created, the overhead increases.
After the initial spike, the time drops sharply. This could happen because the threads start to run concurrently, utilizing system resources more efficiently. The task might be parallelizable, and once enough threads are created it leads to a reduction in the overall time taken.
After the drop, the time starts increasing again as more threads are added. This could be due to:
As the number of threads continues to grow, the CPU has to switch between more threads, leading to increased context switching overhead, which can slow down performance.

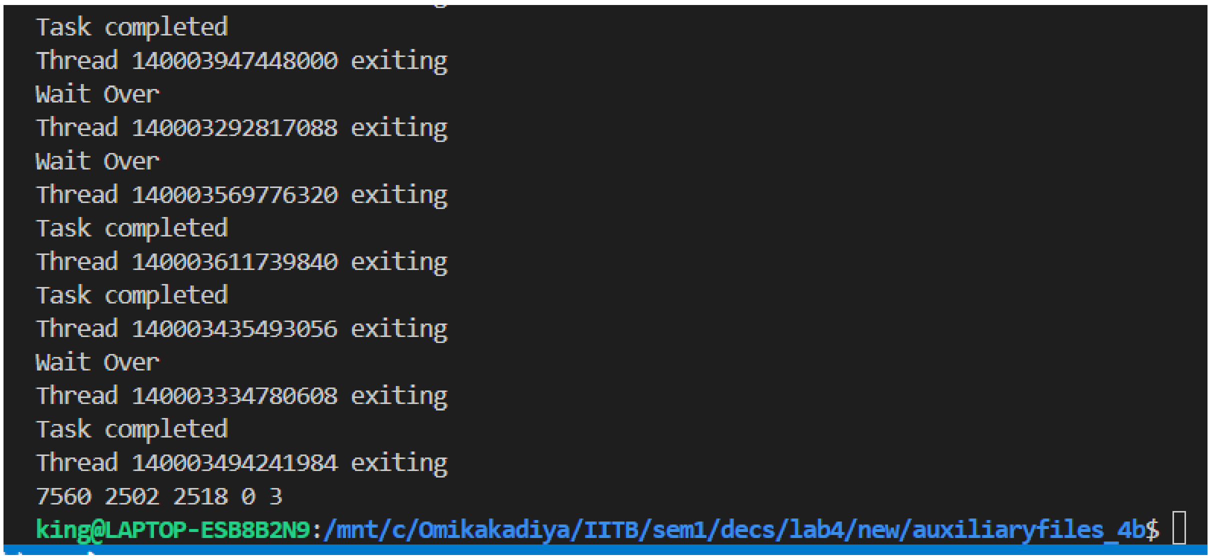
Too many threads can lead to competition for shared resources such as memory, I/O, or CPU time, which again leads to inefficiency.

If the task has been parallelized across all available cores, adding more threads does not improve performance and may actually degrade it as resources are overutilized.

This behavior is common in multithreaded systems where adding threads initially helps with performance but beyond a certain point, the overhead will start begin .


Threads vs Time
Time Taken (in ms)
Number of Threads
3000
2500
2000
1500
1000
500
0
200
400
600
800
1000
1200

2.Taskserver with a threadpool



```
Task completed
Thread 140003947448000 exiting
Wait Over
Thread 140003292817088 exiting
Wait Over
Thread 140003569776320 exiting
Task completed
Thread 140003611739840 exiting
Task completed
Thread 140003435493056 exiting
Wait Over
Thread 140003334780608 exiting
Task completed
Thread 140003494241984 exiting
7560 2502 2518 0 3
king@LAPTOP-ESB8B2N9:/mnt/c/Omikakadiya/IITB/sem1/decs/lab4/new/auxiliaryfiles_4b$
```