CN ASSIGNMENT

Question 1:
The sequential client can be run by ./sequential_client.
Concurrent client can be run by ./concurrent_client.

Question 2:
For running the respective server codes, the commands are as follows:

```
./sequential_server
./fork_server
./thread_server
./select_server
./poll_server
./epoll_server
```

The respective '.txt' files, with intuitive names are created.

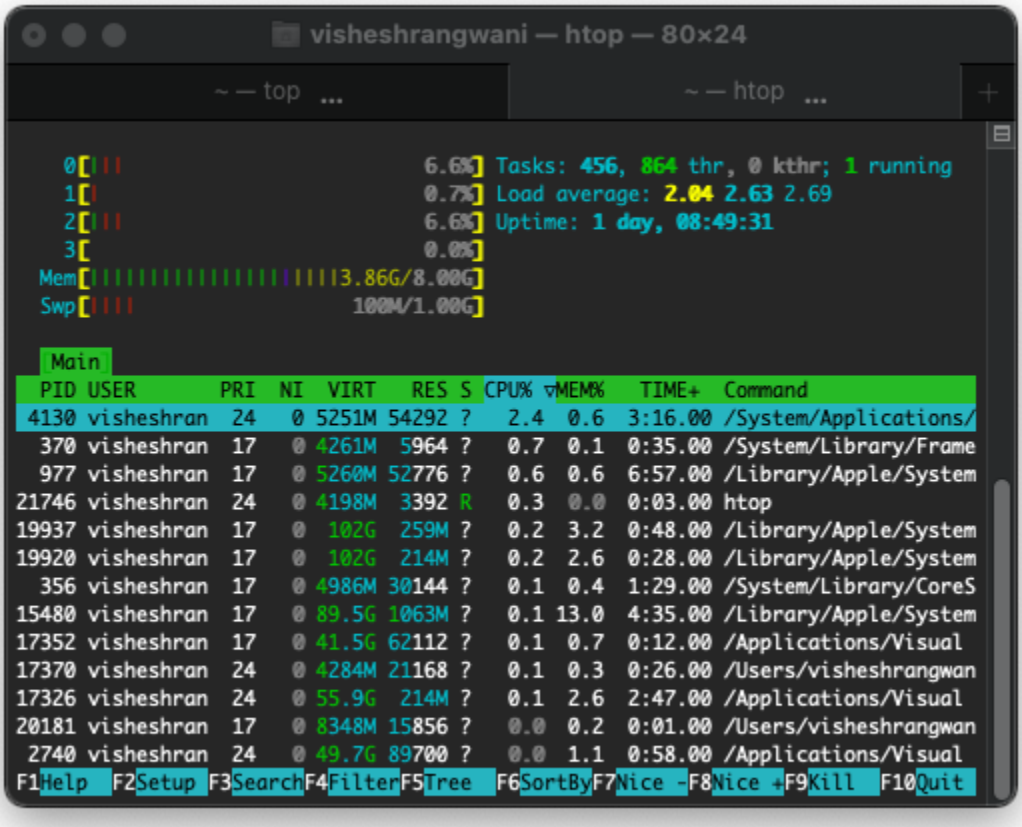
Question 3:

a. Running time for each program is found out using a bash script named script1.sh. It is attached in the folder. For running the concurrent program 10 times using script, use the command: bash script1.sh. This will run 10 concurrent programs simultaneously. But before running this, make sure to comment out all the 'printf()' statements in the 'concurrent_client.c' program, otherwise the time won't be printed. Also comment out the writing part of code in file on each server.

   The observed times for each program are:
   Fork Server: 0.020 seconds
   Thread Server: 0.017 seconds
   Select Server: 0.016 seconds
   Poll Server: 0.015 seconds
   Epoll Server: 0.09 seconds

b. Processor utilization was noted using the htop command when the script was run.

As shown in the attached screenshot. The processor utilization was very low before noting the readings.

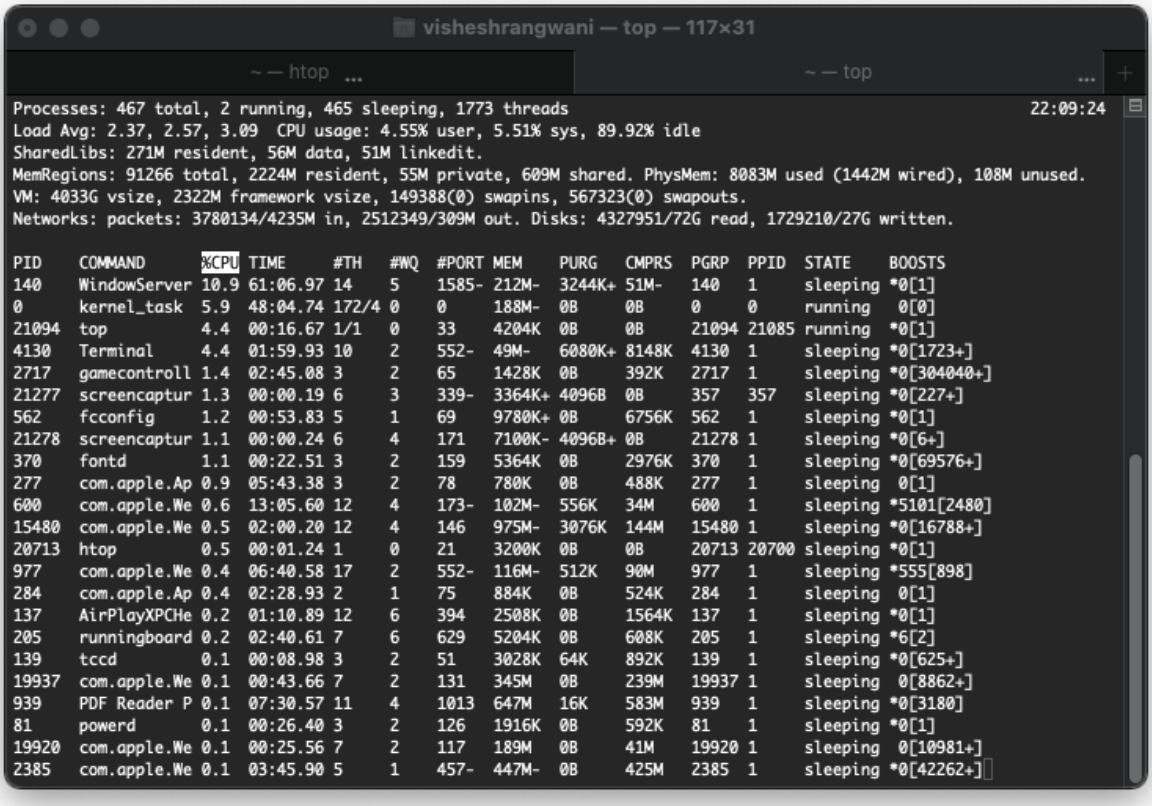


While observing the metrics, the following highest core utilization for different types of servers were:

The observed times for each program are:
Fork Server: 30%
Thread Server: 26%
Select Server: 19%
Poll Server: 18%
Epoll Server: 5%

For the threaded server, the number of threads also increased.

c. Memory used was noted through both top and htop commands. Interface of top command is attached.



The memory increase (about 0.03G) was observed for server using fork while memory increase for multithreaded server was about 0.02 G. In others, memory increase was only 0.01G

Question 4:

Justification for the results in Question 2:

- The program that uses fork takes up maximum memory and CPU utilization because it creates a new process for each client connection. Creating a process requires a lot of resources(memory and processor) as it has to create and maintain a new PCB (Process Control Block) for each client.
- Multithreaded server consumes more resources than the other 3 as the other 3 are event driven programming techniques and are non blocking unlike the multi threaded server which is blocking at ‘accept()’ syscall as well as ‘read()’ syscall.
- In event driven programming using either select, poll or epoll, the file descriptors for each client are monitored and whenever, there is IO, then only blocking syscall ‘read()’ is invoked. This makes the program much more efficient as it doesn’t block at all times, rather keeps track of the file descriptors. There is no ‘busy waiting’ in event driven programming.

- Select is less efficient than poll as it has to consider 3 bitmasks for read, write and exceptions. Poll has just one array for that.
- Poll doesn't have to consider the highest value FD, unlike select.
- Epoll is better than select and poll both as it can monitor the FD in O(1) time whereas poll and select iterate through the entire list of FDs to monitor them, which makes their complexity O(n).
- Also epoll just returns those FDs which are ready for IO. This also makes it use the minimum number of resources.

REFERENCES:

https://devarea.com/linux-io-multiplexing-select-vs-poll-vs-epoll/#.Y0R8Zi8RpQI
Select man pages
Poll man pages
Epoll man pages
Tut slides and videos
https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/
https://www.geeksforgeeks.org/socket-programming-cc/
link
https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
https://suchprogramming.com/epoll-in-3-easy-steps/
https://www.youtube.com/watch?v=dEHZb9JsmOU