

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

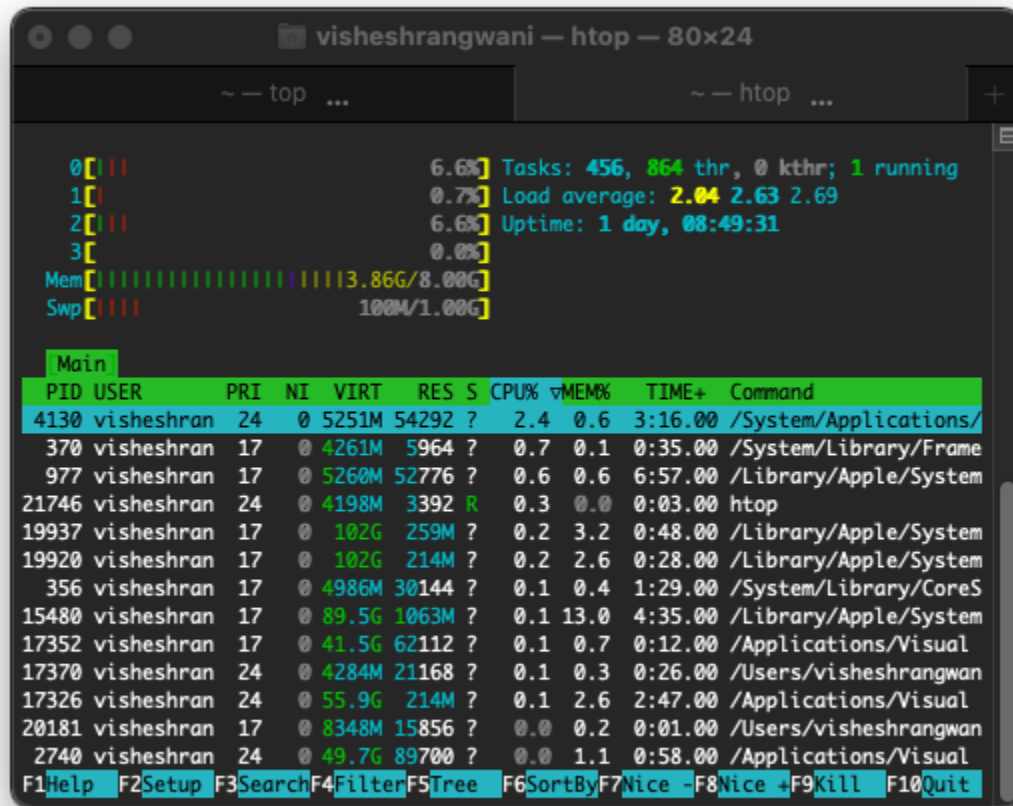
Select Server: 0.016 seconds

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.

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

```
visheshrangwani — top — 117x31
~ — htop ... ~ — top ...
Processes: 467 total, 2 running, 465 sleeping, 1773 threads
Load Avg: 2.37, 2.57, 3.09 CPU usage: 4.55% user, 5.51% sys, 89.92% idle
SharedLibs: 271M resident, 56M data, 51M linkedit.
MemRegions: 91266 total, 2224M resident, 55M private, 609M shared. PhysMem: 8083M used (1442M wired), 108M unused.
VM: 4033G vsize, 2322M framework vsize, 149388(0) swapins, 567323(0) swapouts.
Networks: packets: 3780134/4235M in, 2512349/309M out. Disks: 4327951/72G read, 1729210/27G written.

PID  COMMAND    %CPU  TIME    #TH  #WQ  #PORT  MEM    PURG  CMPRS  PGRP  PPID  STATE  BOOSTS
140  WindowServer 10.9  61:06.97 14   5   1585- 212M- 3244K+ 51M- 140   1   sleeping *0[1]
0    kernel_task  5.9   48:04.74 172/4 0     0    188M- 0B    0      0    0    running 0[0]
21094 top        4.4   00:16.67 1/1   0    33    4204K 0B    0      21094 21085 running *0[1]
4130  Terminal    4.4   01:59.93 10    2   552- 49M-  6080K+ 8148K 4130 1   sleeping *0[1723+]
2717  gamecontrol 1.4   02:45.08 3     2    65    1428K 0B    392K   2717 1   sleeping *0[304040+]
21277 screencaptur 1.3   00:00.19 6     3   339- 3364K+ 4096B 0B    357   357   sleeping *0[227+]
562   fcconfig    1.2   00:53.83 5     1    69    9780K+ 0B    6756K 562   1   sleeping *0[1]
21278 screencaptur 1.1   00:00.24 6     4   171    7100K- 4096B+ 0B    21278 1   sleeping *0[6+]
370   fontd       1.1   00:22.51 3     2   159    5364K 0B    2976K 370   1   sleeping *0[69576+]
277   com.apple.Ap 0.9   05:43.38 3     2    78    780K 0B    488K   277   1   sleeping 0[1]
600   com.apple.We 0.6   13:05.60 12    4   173- 102M- 556K   34M   600   1   sleeping *5101[2480]
15480 com.apple.We 0.5   02:00.20 12    4   146    975M- 3076K 144M  15480 1   sleeping *0[16788+]
20713 htop        0.5   00:01.24 1     0    21    3200K 0B    0      20713 20700 sleeping *0[1]
977   com.apple.We 0.4   06:40.58 17    2   552- 116M- 512K   90M   977   1   sleeping *555[898]
284   com.apple.Ap 0.4   02:28.93 2     1    75    884K 0B    524K   284   1   sleeping 0[1]
137   AirPlayXPCh 0.2   01:10.89 12    6   394    2508K 0B    1564K 137   1   sleeping *0[1]
205   runningboard 0.2   02:40.61 7     6   629    5204K 0B    608K   205   1   sleeping *6[2]
139   tccd        0.1   00:08.98 3     2    51    3028K 64K    892K   139   1   sleeping *0[625+]
19937 com.apple.We 0.1   00:43.66 7     2   131    345M 0B    239M   19937 1   sleeping 0[8862+]
939   PDF Reader P 0.1   07:30.57 11    4   1013  647M 16K    583M   939   1   sleeping *0[3180]
81    powerd      0.1   00:26.40 3     2   126    1916K 0B    592K   81    1   sleeping *0[1]
19920 com.apple.We 0.1   00:25.56 7     2   117    189M 0B    41M    19920 1   sleeping 0[10981+]
2385  com.apple.We 0.1   03:45.90 5     1   457- 447M- 0B    425M   2385 1   sleeping *0[42262+]
```

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>