CN Assignment 3 Report

Desktop Configuration: 6 GB RAM and 8 Cores for Benchmarking

There are a total of 5 different configurations of server with 3 different combinations of concurrent clients. That are:

- Blocking
    1. Fork
    2. Thread

- Non Blocking
    1. Select
    2. Poll
    3. Epoll

The three different combinations of concurrent clients are 500, 1000 and 3000.

Note:
Select API calls only support up to 1024 concurrent connections. Hence only 500, 1000 are complete. For 3000, all the rest of the packets are dropped.

We must set ulimit -n 8192 on both the server and the client in order to support the 3000 concurrent connections.

Method for using the time command:
/usr/bin/time -v./server and hit Ctrl+C after the client has finished receiving

Method for using the htop command:
On the server, open two terminals. Run the server on one terminal, then use the second terminal to the command htop.

For sending and receiving:

a. Read the payload and cast it to a 64-bit unsigned integer, n.
b. Compute the factorial of the “n”; if n>20, then calculate the factorial of 20.
c. Send the factorial result as the response message back to the client.

Server Program	CPU Utilization (%)	Memory Utilization (MB)
Poll 500	29.3	1.66
Poll 1000	58.5	1.78
Poll 3000	84.0	1.78
Epoll 500	7.92	1.66
Epoll 1000	42.3	1.66
Epoll 3000	71.5	1.66
Select 500	54.1	1.66
Select 1000	92.7	1.66
Fork 500	2.8	1.40
Fork 1000	6.7	1.40
Fork 3000	11.4	1.40
Thread 500	39.1	5.62
Thread 1000	66.9	2.20
Thread 3000	140.1%	6.30

Analysis of HTOP and Time Command

Memory Analysis

epoll and poll efficiency: Both epoll and poll show consistent memory usage regardless of the number of file descriptors, contradicting the common belief that poll is less memory efficient than epoll at scale.

select memory usage: Select maintains a constant memory usage across different file descriptor configurations, indicating it does not require additional memory for additional file descriptors in the tested scenario.

fork advantages: Forking processes demonstrate the lowest memory usage, likely due to the use of copy-on-write for shared memory pages, or it could suggest that each process has a relatively small memory footprint.

thread memory scaling: Threads exhibit a notable increase in memory usage with additional file descriptors, as each thread has its own stack and potentially other thread-local storage, leading to a substantial memory footprint with more threads.

Non-linear memory scaling: Memory usage for epoll and poll does not increase linearly with the number of file descriptors, showcasing effective memory management for these mechanisms in the given test cases.

Caveat with fork configurations: The memory usage for fork configurations may not account for the total memory footprint if shared pages are counted multiple times across processes.

Concerns with thread configurations: The pattern observed with threads indicates that more threads result in significantly more memory usage, potentially posing challenges at scale due to increased memory demands.

CPU Analysis

epoll efficiency: epoll_500 showcases the lowest CPU usage, indicating that managing 500 file descriptors with epoll is not excessively CPU-intensive.

CPU scaling with file descriptors: CPU usage increases notably as the number of file descriptors grows in epoll, reflecting the expected correlation between CPU utilization and the number of monitored descriptors.

poll performance: poll_500 exhibits extremely low CPU usage, but it rises significantly as the number of descriptors increases, although not as high as epoll for poll_3000.

select efficiency: Select demonstrates moderate CPU usage for 500 file descriptors but becomes significantly higher with 1000 descriptors, suggesting that select's scalability might not be as efficient as epoll when the number of monitored descriptors increases.

fork advantages: Forking processes consistently display the lowest CPU usage across all configurations for 500 and 1000 file descriptors, with a slight increase for 3000 descriptors. This suggests that the process creation overhead is not excessively CPU-intensive or that the created processes are not performing intensive computations.

Thread behavior: Thread configurations indicate moderate CPU usage for 500 and 1000 descriptors, but exceptionally high usage for 3000 descriptors, potentially due to the overhead of managing a large number of threads, which can be more CPU-intensive compared to handling the same number of file descriptors with epoll or poll.

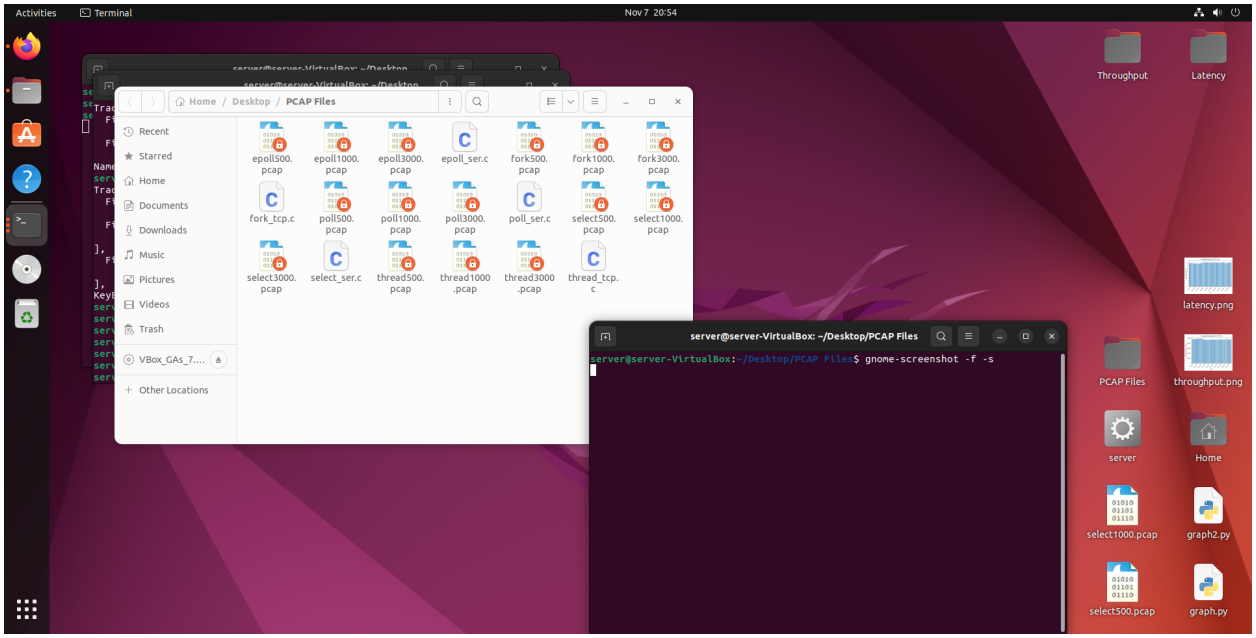
Comparative performance: epoll consistently outperforms select and poll with a higher number of file descriptors, aligning with the general understanding of epoll's efficiency in managing multiple file descriptors.

Insights on fork configuration: The consistently low CPU usage in the fork configuration may indicate a workload that involves minimal computation after forking.

Thread considerations: Thread configurations highlight that threading can be more CPU-intensive at higher scales, likely due to the overhead of context switching and synchronization among threads.

Analysis of the graphs of the throughput and latencies

There is a common tendency that shows that throughput drops and latencies rise with an increase in the number of concurrent connections. The reason for this is because as the number of connections rises, so does the server's workload. This causes a lot of packets to be lost, requiring the client to retransmit them or send them again. This is further visible in each pcap file's exchanged packet count. Throughput therefore drops. Additionally, latencies rise for every flow for comparable reasons. Think about latencies as the interval of time between each flow's initial and last packet. This will rise in tandem with the number of lost packets that require retransmission. The time interval between the first and last packet will grow as the number of retransmissions rises. Plots that I have supplied also demonstrate this.


Activities
Terminal
Nov 7 20:54
Throughput
Latency
Home / Desktop / PCAP Files
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
VBox_GAs_7....
Other Locations
epoll500.pcap
epoll1000.pcap
epoll3000.pcap
epoll_ser.c
fork500.pcap
fork1000.pcap
fork3000.pcap
fork_tcp.c
poll500.pcap
poll1000.pcap
poll3000.pcap
poll_ser.c
select500.pcap
select1000.pcap
select3000.pcap
select_ser.c
thread500.pcap
thread1000.pcap
thread3000.pcap
thread_tcp.c
server@server-VirtualBox: ~/Desktop/PCAP Files
server@server-VirtualBox:~/Desktop/PCAP Files$ gnome-screenshot -f -s
latency.png
PCAP Files
throughput.png
server
Home
select1000.pcap
graph2.py
select500.pcap
graph.py