CSE 232: Programming Assignment 2
Socket programming with performance analysis

Anushka Srivastava (2022086) Apaar (2022089)

September 2024

1 Performance Analysis of different TCP servers

1.1 CPU Usage

a. Single Threaded TCP Client-Server

```
Performance counter stats for './server':

            3.62 msec cpu-clock                 #    0.000 CPUs utilized
            3.63 msec task-clock                #    0.000 CPUs utilized
        12824855      cycles                    #    3.536 GHz
        10465723      instructions              #    0.82  insn per cycle
              16      context-switches          #    4.412 K/sec

     14.854013778 seconds time elapsed

      0.000000000 seconds user
      0.005054000 seconds sys
```

Figure 1: CPU Usage Metrics for Single Threaded TCP Client-Server

b. Concurrent TCP client-server

```
Performance counter stats for './server':

            8.55 msec cpu-clock                 #    0.000 CPUs utilized
            8.56 msec task-clock                #    0.000 CPUs utilized
        15980522      cycles                    #    1.868 GHz
        11400701      instructions              #    0.71  insn per cycle
              28      context-switches          #    3.273 K/sec

     50.712897980 seconds time elapsed

      0.008286000 seconds user
      0.000000000 seconds sys
```

Figure 2: CPU Usage Metrics for Concurrent TCP client-server

c. TCP client-server using “select”

```
Performance counter stats for './server':

            3.63 msec cpu-clock                 #    0.000 CPUs utilized
            3.63 msec task-clock                #    0.000 CPUs utilized
        11810191      cycles                    #    3.255 GHz
        11010260      instructions              #    0.93  insn per cycle
              12      context-switches          #    3.307 K/sec

      8.091410237 seconds time elapsed

      0.000000000 seconds user
      0.004843000 seconds sys
```

Figure 3: CPU Usage Metrics for TCP client-server using “select”

We will create plots to compare the different metrics of the 3 server models with each other.

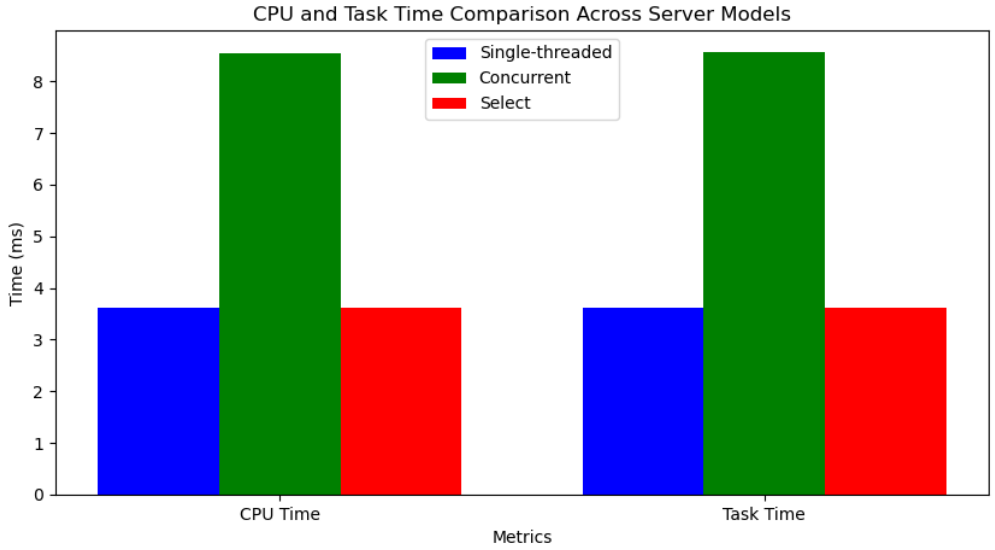

Figure 4: CPU and Task Time Comparison Across Server Models

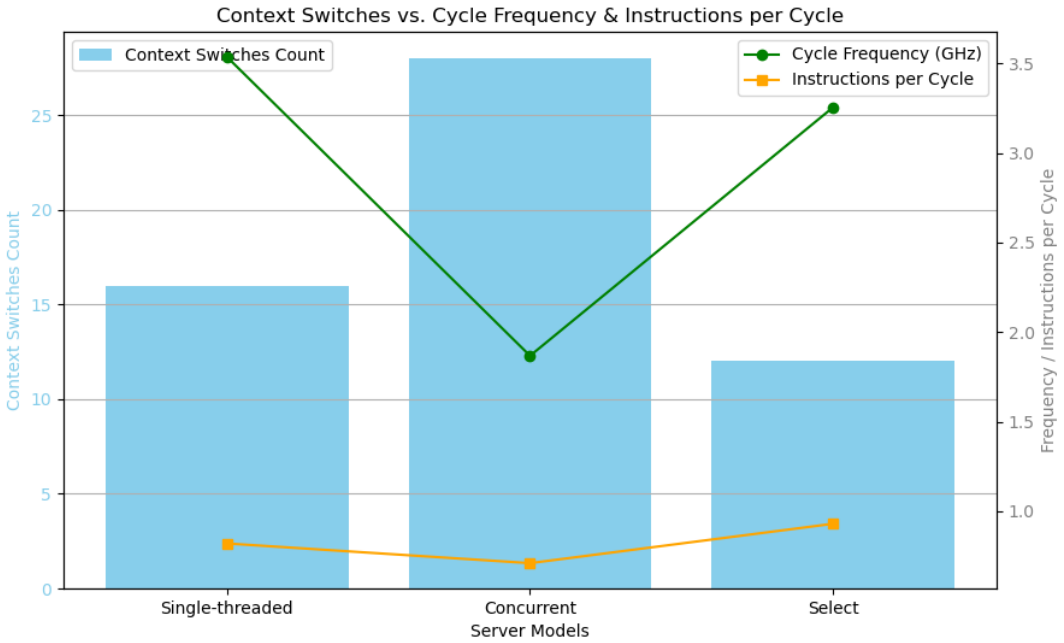

Figure 5: Context Switches vs. Cycle Frequency & Instructions per Cycle

Analysis

CPU Clock Time

- The **Concurrent** model has the highest CPU clock time, indicating that it consumes more CPU cycles to handle concurrent tasks. This could be due to the overhead of managing multiple threads, increased context switches, and resource sharing.
- The **Single Threaded** model has the lowest CPU time, meaning fewer CPU cycles are used. This model operates serially, avoiding the overhead of context switching, making it more CPU-efficient per task.
- The **Select** model lies between the two in terms of CPU clock cycles but it is more comparable to the single threaded model.

Task Clock Time

- The **Concurrent** model has the highest task clock time, implying that despite using more CPU resources, it takes longer to complete tasks. This is likely due to the overhead caused by managing multiple threads and context switches.
- The **Single Threaded** model has the shortest task clock time, meaning tasks complete faster in wall-clock time, but this model sacrifices parallelism.
- The **Select** model lies between the two in terms of task clock cycles but it is more comparable to the single threaded model.

Context Switches

- The **Concurrent** model has the highest number of context switches. This directly impacts CPU clock time, as context switching between threads requires saving and restoring state, which increases CPU cycle usage and contributes to higher task clock time.
- The **Single Threaded** model has fewer context switches, which helps reduce CPU clock time, as there is less overhead involved in task scheduling.
- The **Select** model is more comparable to the single-threaded model.

Cycle Frequency
The frequency of CPU cycles seems to increase slightly for the Concurrent and Select models, possibly due to more complex processing demands.

Instructions per Cycle
There's minimal change across all models for instructions per cycle, indicating that all server models are relatively consistent in terms of instruction efficiency.

We now increase the number of clients from 5 to 15 and analyse the CPU usage for the same.

a. Single Threaded TCP Client-Server

```
Performance counter stats for './server':

            9.77 msec cpu-clock                 #    0.001 CPUs utilized
            9.77 msec task-clock                #    0.001 CPUs utilized
        36040763      cycles                    #    3.688 GHz
        29906703      instructions              #    0.83  insn per cycle
              32      context-switches          #    3.275 K/sec

      8.429334519 seconds time elapsed

      0.005439000 seconds user
      0.004902000 seconds sys
```

Figure 6: CPU Usage Metrics for Single Threaded TCP Client-Server

b. Concurrent TCP client-server

```
Performance counter stats for './server':

            9.49 msec cpu-clock                 #    0.001 CPUs utilized
            9.50 msec task-clock                #    0.001 CPUs utilized
        30173527      cycles                    #    3.177 GHz
        31641235      instructions              #    1.05  insn per cycle
              61      context-switches          #    6.422 K/sec

      18.236710414 seconds time elapsed

       0.007051000 seconds user
       0.002073000 seconds sys
```

Figure 7: CPU Usage Metrics for Concurrent TCP client-server

c. TCP client-server using "select"

```
Performance counter stats for './server':

            8.62 msec cpu-clock                 #    0.001 CPUs utilized
            8.62 msec task-clock                #    0.001 CPUs utilized
        29458381      cycles                    #    3.418 GHz
        30117072      instructions              #    1.02  insn per cycle
              12      context-switches          #    1.392 K/sec

      8.286280893 seconds time elapsed

      0.004752000 seconds user
      0.004752000 seconds sys
```

Figure 8: CPU Usage Metrics for TCP client-server using “select”

We will create plots to compare the different metrics of the 3 server models with each other.

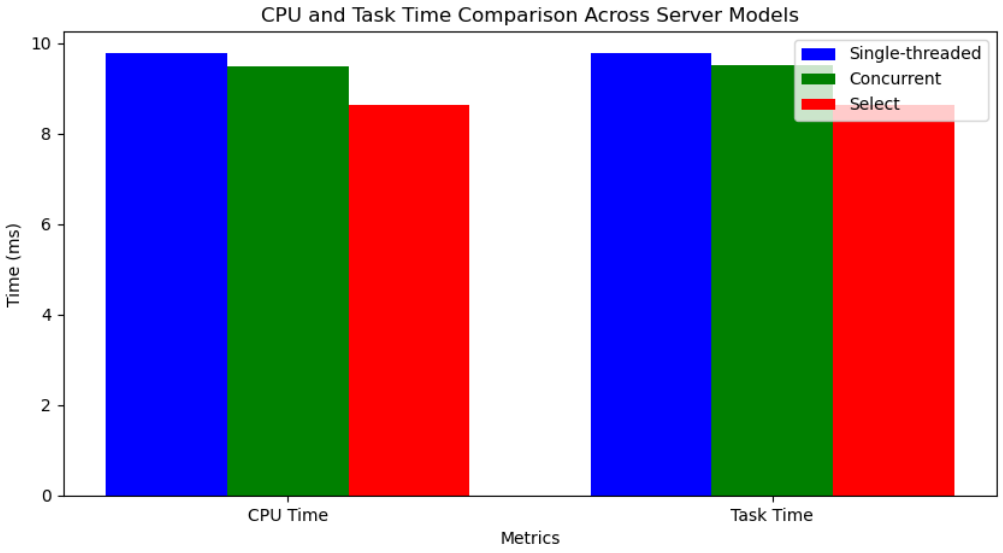

Figure 9: CPU and Task Time Comparison Across Server Models

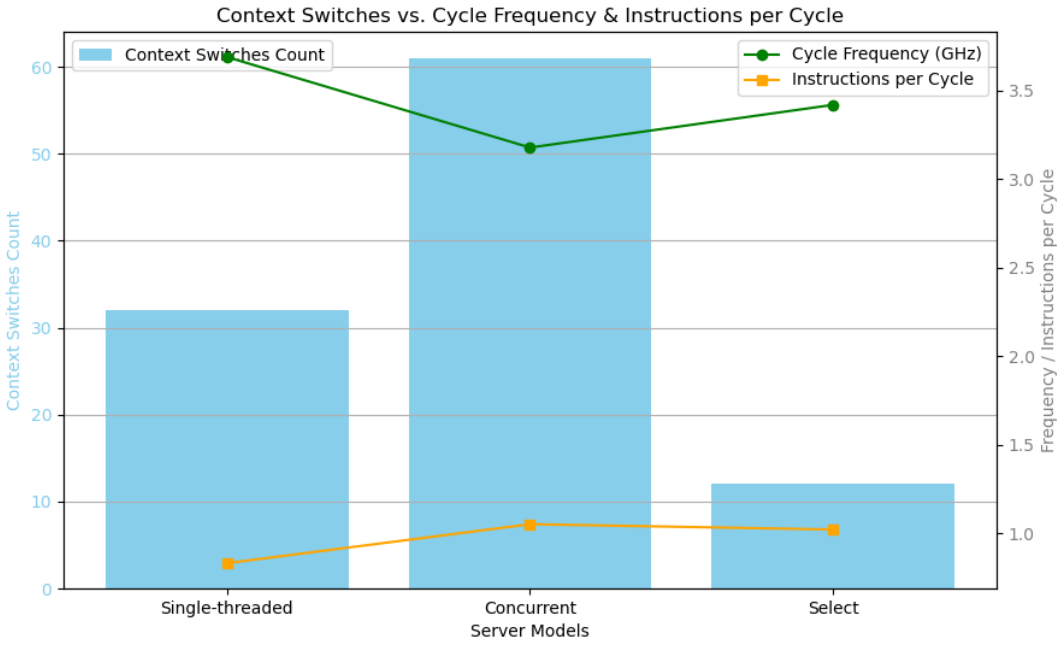

Figure 10: Context Switches vs. Cycle Frequency & Instructions per Cycle

Analysis

We see that the difference in the server performance becomes clearer when we increase the number of clients.

CPU Clock Time

- The **Concurrent** model has the highest CPU clock time, as managing threads or processes for multiple clients involves more overhead in terms of CPU usage and resource management, especially with many context switches.
- The **Single Threaded** model is lower in CPU time than the Concurrent model and is less efficient than the Select model because it can only handle one task at a time.
- The **Select** model has the lowest CPU clock time, indicating that it is more efficient at managing tasks compared to both the Single-threaded and Concurrent models. This lower CPU time is likely due to its event-driven nature, where it can handle multiple clients with fewer resource-heavy operations (like thread creation and management), reducing the overall CPU load.

Task Clock Time

- The **Concurrent** model has the highest task clock time, as managing multiple threads and context switches adds to the total time needed to complete tasks, even though it's theoretically parallelizing work.
- The **Single Threaded** model lies in middle of both the server with respect to task clock time.
- The **Select** model has the lowest task time, meaning tasks are completed faster with fewer delays or overhead.

Context Switches

- The **Concurrent** model has the highest number of context switches. With more threads or processes being created and switched between for multiple clients, this overhead becomes substantial.
- The **Single Threaded** model has more context switches than the Select model but fewer than the Concurrent model, as it doesn't need to switch between threads but still handles fewer clients and tasks sequentially.

- The **Select** model has the fewest context switches. This is because its event-driven architecture doesn't rely on creating new threads or processes for each client.

Inference

We observe that as we increase the number of clients, the **TCP model using "select"** is the most efficient in terms of CPU usage. This model scales well as the number of clients increases because of its event-driven, non-blocking I/O architecture, which avoids the overhead associated with thread management in the Concurrent model. It is well-suited for handling high numbers of clients with minimal resource consumption.

Although concurrent model offers parallelism, it comes at the cost of CPU cost.

The Single-threaded model remains efficient only for a small number of clients, with moderate context switches and clock times. However, it doesn't scale well as client numbers increase because it can only handle tasks one at a time, creating a bottleneck for larger workloads.

1.2 Memory and Cache

a. Single Threaded TCP Client-Server

```
Performance counter stats for './server':

          439132      cache-references
          123883      cache-misses              #   28.211 % of all cache refs

      7.015646779 seconds time elapsed

      0.003176000 seconds user
      0.000000000 seconds sys
```

Figure 11: CPU Usage Metrics for Single Threaded TCP Client-Server

b. Concurrent TCP client-server

```
Performance counter stats for './server':

          581786      cache-references
          210215      cache-misses              #   36.133 % of all cache refs

     99.574054678 seconds time elapsed

      0.006730000 seconds user
      0.000000000 seconds sys
```

Figure 12: CPU Usage Metrics for Concurrent TCP client-server

c. TCP client-server using "select"

```
Performance counter stats for './server':

          444674      cache-references
          126795      cache-misses              #   28.514 % of all cache refs

    15.609013854 seconds time elapsed

     0.004112000 seconds user
     0.000000000 seconds sys
```

Figure 13: CPU Usage Metrics for TCP client-server using "select"

Now, we create plots to analyse the results.

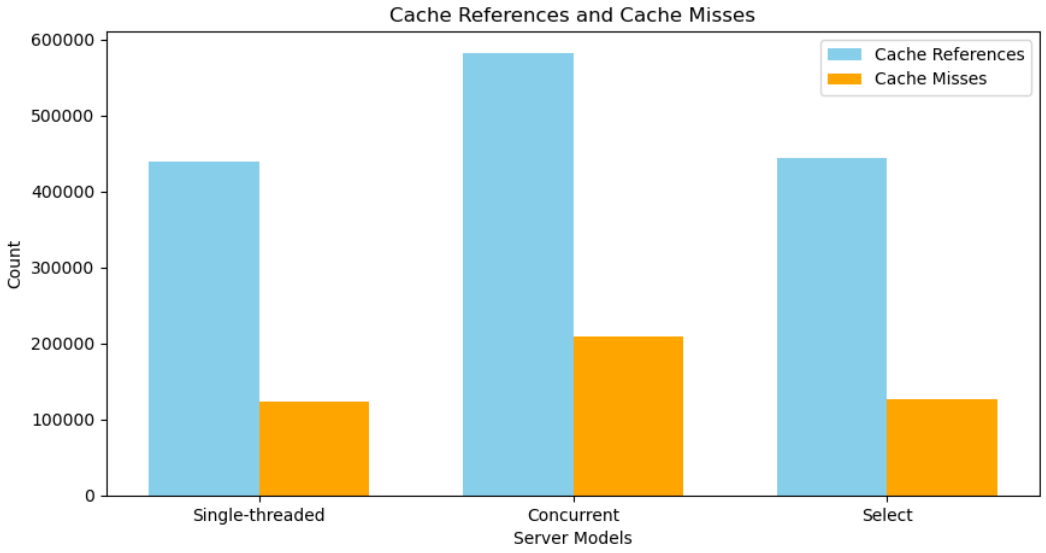

Figure 14: Cache References and Cache Misses

Analysis

1. **Concurrent model** has the highest number of cache references, which indicates that it accesses the CPU cache more frequently. This is likely due to its ability to handle multiple tasks simultaneously, causing it to utilize more memory and thus reference the cache more often. It also has the highest number of cache misses, likely due to the overhead of handling many threads and processes, each of which needs its own data in the cache.

2. The **Single Threaded model** and the **Select model** has comparable cache references and cache misses, both of which are lower than the concurrent model, as they have no thread to process.

We now increase the number of clients from 5 to 15 and analyse the CPU usage for the same.

a. Single Threaded TCP Client-Server

```
Performance counter stats for './server':

          1164705      cache-references
           352507      cache-misses              #   30.266 % of all cache refs

       8.088391103 seconds time elapsed

       0.005777000 seconds user
       0.005076000 seconds sys
```

Figure 15: CPU Usage Metrics for Single Threaded TCP Client-Server

b. Concurrent TCP client-server

```
Performance counter stats for './server':

          1477448      cache-references
           413584      cache-misses              #   27.993 % of all cache refs

       9.164791972 seconds time elapsed

       0.006811000 seconds user
       0.010026000 seconds sys
```

Figure 16: CPU Usage Metrics for Concurrent TCP client-server

c. TCP client-server using "select"

```
Performance counter stats for './server':

          1125154      cache-references
           286367      cache-misses              #   25.451 % of all cache refs

       9.022424170 seconds time elapsed

       0.001513000 seconds user
       0.012584000 seconds sys
```

Figure 17: CPU Usage Metrics for TCP client-server using "select"

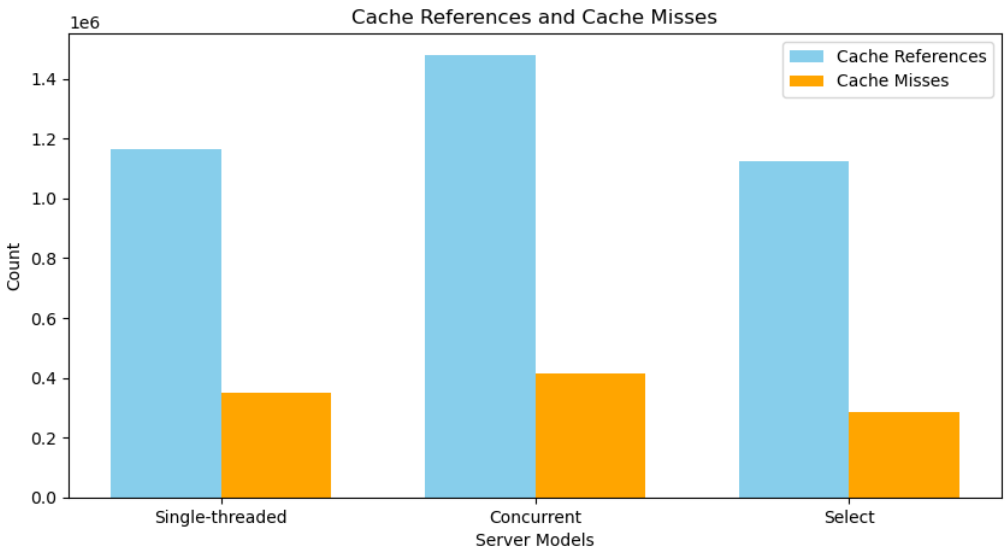

Figure 18: Cache References and Cache Misses

Analysis

On increasing the number of clients, we see similar results to when the number of clients was 5. It is visible that the number of cache-misses and cache-references is lower for the **Select model** than the **Single Threaded model**.

Inference

The **Concurrent model** handles the most tasks but at the cost of heavy cache usage and inefficiency due to frequent cache misses, especially as the number of clients increases.

The **Single Threaded model** offers a good balance between cache usage and cache efficiency, but its single-task processing nature limits its scalability.

The **Select model** is the most cache-efficient, with the lowest number of cache references and misses, handling multiple clients with fewer cache misses.

2 Conclusion

The best TCP client-server model can be decided on the basis of our requirements and availability of resources.

- A **Concurrent TCP model** offers parallelism and utilizes multi-core CPU processors efficiently. However, it has higher memory overhead and CPU clock time due to larger number of threads and is more complex. For CPU-intensive tasks or to fully utilize multi-core processors where we can afford higher overheads, a concurrent server could be better.
- A **Single Threaded TCP model** is best for handling low number of clients, where concurrency is not a requirement. It has lower overheads in term of memory and context switching. However, it has a comparatively poorer performance with many simultaneous connections or long-running operations. For a small number of clients or simple applications, a single-threaded server might be sufficient.
- A **TCP model implemented using Select** is efficient for handling many connections with lower overheads. Since it is event-driven non-blocking I/O architecture, it avoids overheads associated with thread management in the Concurrent model. It is well-suited for handling high numbers of clients with minimal resource consumption.

References

[1] Github: Socket Programming

[2] Stack Overflow: Get Process Info from Proc

[3] Stack Overflow: Setting perf on WSL

[4] Coding Bison: Socket Programming: Socket Select

[5] Stack Overflow: Destroying a Socket Connection in C

[6] Dev.to: Single-Threaded vs Multi-Threaded Servers