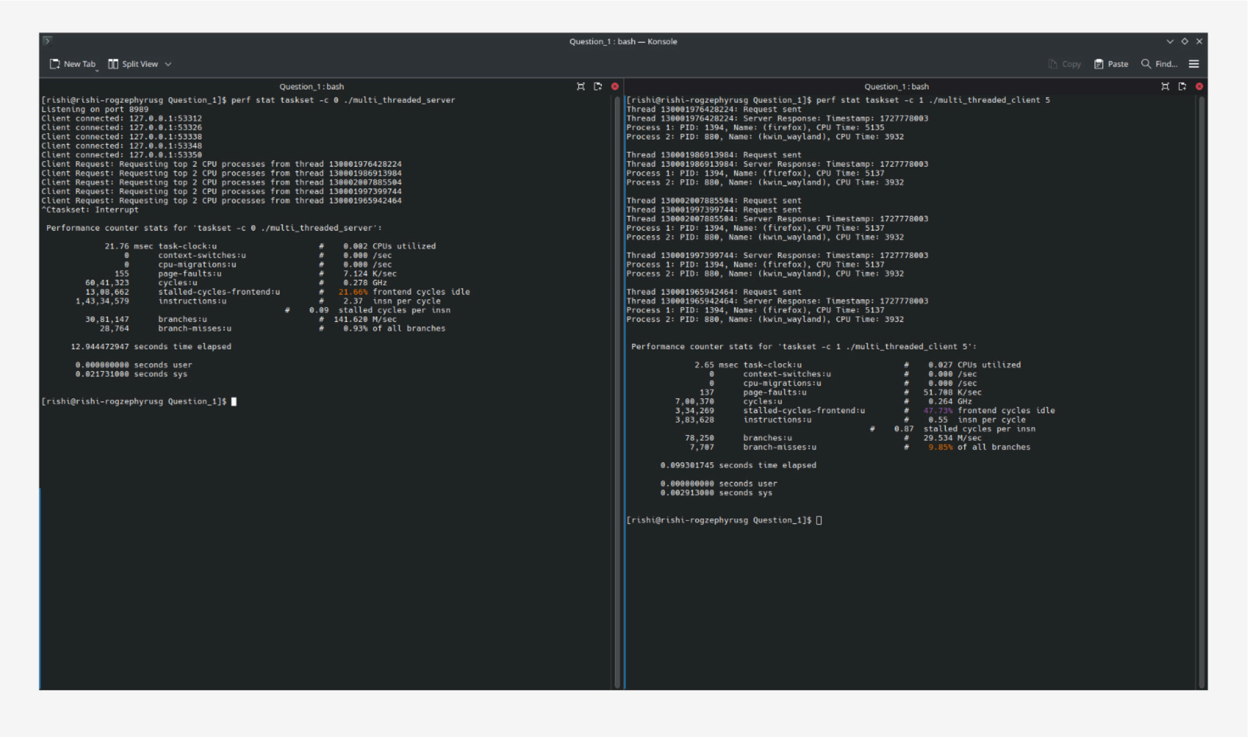
CN Assignment 2

Rishi Pendyala - 2022403
Rishit - 2022405
github

Q.1. Write a client-server socket program in C. The client process and the server process should run on separate VMs (or containers) and communicate with each other. Use "taskset" to pin the process to specific CPUs. This helps measure the performance. Your program should have the following features. [1+(1+2)+2+(1+2)+1+1]

1. The server sets up a TCP socket and listens for client connections.
2. The server should be able to handle multiple concurrent clients (multithreaded, concurrent server). The server accepts the client connection; hence, a new socket is created with 4 tuples (server IP, server listening port, client IP, client port). The server creates a new thread that continues to process the client connection (Hint: Use pthread library for multithreading). The original server socket continues to listen on the same listening port for newer incoming client connections.
3. The client creates a socket and initiates the TCP connection. Your client process should support initiating "n" concurrent client connection requests, where "n" is passed as a program argument.
4. After the client connection is established, the client sends a request to the server to get information about the server's top TWO CPU-consuming processes. The server finds out the top CPU-consuming process (user+kernel CPU time) and gathers information such as process name, pid (process id), and CPU usage of the process in user & kernel mode (you can report this time in clock ticks).
    ○ Possible solution approach: The server should make use of the open() system call to read the proc" filesystem to get this information. You need to read /proc/[pid]/stat for all processes to parse the process name, pid, and CPU time (user+kernel). To understand the format of /proc/[pid]/stat file, refer the source code of show_stat function is available here for reference. More about "proc" file system is here.
5. Server sends the information collected in step (4) to the client.
6. The client prints this information & closes the connection.



```
[rishi@rishi-rogzephyrusg Question_1]$ perf stat taskset -c 0 ./multi_threaded_server
Listening on port 8989
Client connected: 127.0.0.1:53312
Client connected: 127.0.0.1:53326
Client connected: 127.0.0.1:53338
Client connected: 127.0.0.1:53348
Client connected: 127.0.0.1:53350
Client Request: Requesting top 2 CPU processes from thread 130001976428224
Client Request: Requesting top 2 CPU processes from thread 130001986913984
Client Request: Requesting top 2 CPU processes from thread 130002007885504
Client Request: Requesting top 2 CPU processes from thread 130001997399744
Client Request: Requesting top 2 CPU processes from thread 130001965942464
^Ctaskset: Interrupt

 Performance counter stats for 'taskset -c 0 ./multi_threaded_server':

             21.76 msec task-clock:u                     #    0.002 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               155      page-faults:u                    #    7.124 K/sec
         60,41,323      cycles:u                         #    0.278 GHz
         13,08,662      stalled-cycles-frontend:u        #   21.66% frontend cycles idle
       1,43,34,579      instructions:u                   #    2.37  insn per cycle
                                                  #    0.09  stalled cycles per insn
         30,81,147      branches:u                       #  141.620 M/sec
            28,764      branch-misses:u                  #    0.93% of all branches

      12.944472947 seconds time elapsed

       0.000000000 seconds user
       0.021731000 seconds sys


[rishi@rishi-rogzephyrusg Question_1]$ █
```

```
[rishi@rishi-rogzephyrusg Question_1]$ perf stat taskset -c 1 ./multi_threaded_client 5
Thread 130001976428224: Request sent
Thread 130001976428224: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5135
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001986913984: Request sent
Thread 130001986913984: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130002007885504: Request sent
Thread 130001997399744: Request sent
Thread 130002007885504: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001997399744: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001965942464: Request sent
Thread 130001965942464: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932


 Performance counter stats for 'taskset -c 1 ./multi_threaded_client 5':

              2.65 msec task-clock:u                     #    0.027 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               137      page-faults:u                    #   51.708 K/sec
          7,00,370      cycles:u                         #    0.264 GHz
          3,34,269      stalled-cycles-frontend:u        #   47.73% frontend cycles idle
          3,83,628      instructions:u                   #    0.55  insn per cycle
                                                  #    0.87  stalled cycles per insn
            78,250      branches:u                       #   29.534 M/sec
             7,707      branch-misses:u                  #    9.85% of all branches

       0.099301745 seconds time elapsed

       0.000000000 seconds user
       0.002913000 seconds sys


[rishi@rishi-rogzephyrusg Question_1]$ ▯
```

Q.2. The socket programming source code that leverages “select” system call here. Modify the server code as per Q.1. Use the perf tool to analyze the performance of the following: [3+3+3]

(a) Single-threaded TCP client-server

(b) Concurrent TCP client-server

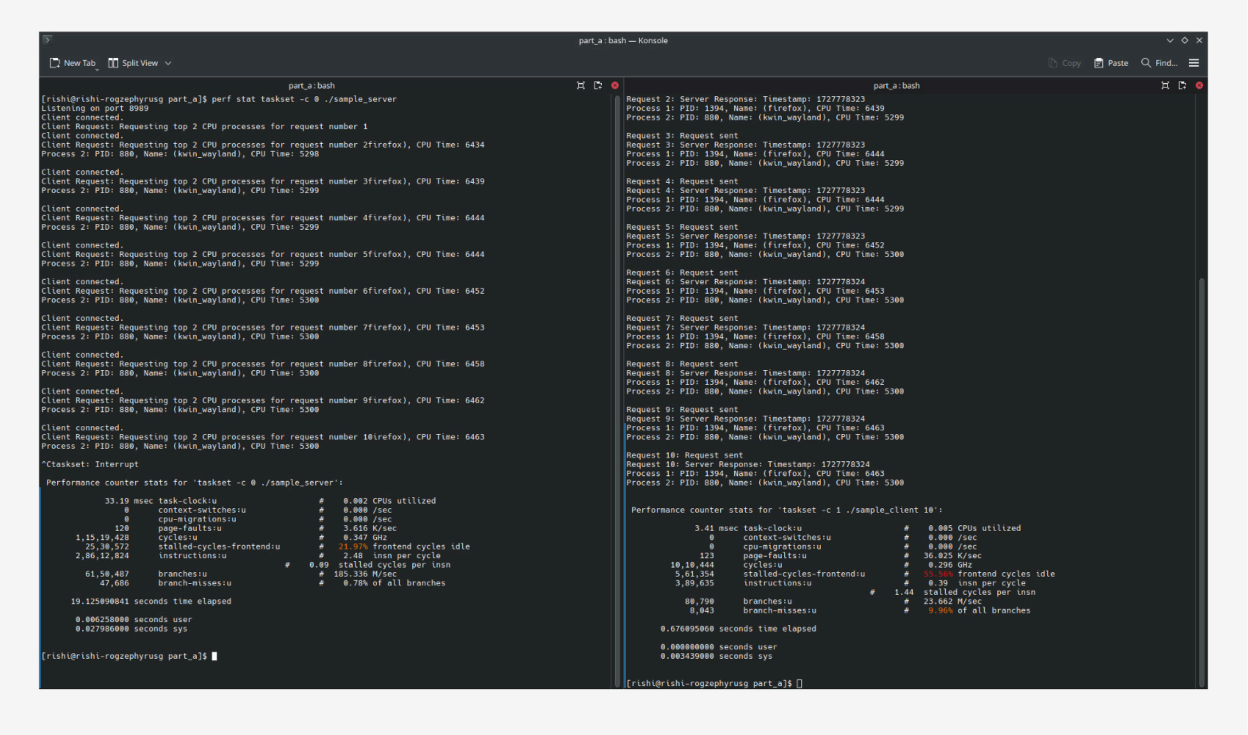
(c) TCP client-server using “select”

You can be creative for this analysis. You may take readings of various performance counters (CPU clocks, cache misses, context switches, etc.) across number of concurrent connections, etc. This is an OPEN question; you can be as comprehensive as you can.

Single-threaded TCP client-server

Single-threaded TCP client-server

(a) Single-threaded TCP client-server



```
[rishi@rishi-rogzephyrusg part_a]$ perf stat taskset -c 0 ./sample_server
Listening on port 8989
Client connected.
Client Request: Requesting top 2 CPU processes for request number 1
Client connected.
Client Request: Requesting top 2 CPU processes for request number 2firefox), CPU Time: 6434
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5298

Client connected.
Client Request: Requesting top 2 CPU processes for request number 3firefox), CPU Time: 6439
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Client connected.
Client Request: Requesting top 2 CPU processes for request number 4firefox), CPU Time: 6444
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Client connected.
Client Request: Requesting top 2 CPU processes for request number 5firefox), CPU Time: 6444
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Client connected.
Client Request: Requesting top 2 CPU processes for request number 6firefox), CPU Time: 6452
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Client connected.
Client Request: Requesting top 2 CPU processes for request number 7firefox), CPU Time: 6453
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Client connected.
Client Request: Requesting top 2 CPU processes for request number 8firefox), CPU Time: 6458
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Client connected.
Client Request: Requesting top 2 CPU processes for request number 9firefox), CPU Time: 6462
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Client connected.
Client Request: Requesting top 2 CPU processes for request number 10irefox), CPU Time: 6463
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

^Ctaskset: Interrupt

 Performance counter stats for 'taskset -c 0 ./sample_server':

             33.19 msec task-clock:u                     #    0.002 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               120      page-faults:u                    #    3.616 K/sec
       1,15,19,428      cycles:u                         #    0.347 GHz
         25,30,572      stalled-cycles-frontend:u        #   21.97% frontend cycles idle
       2,86,12,824      instructions:u                   #    2.48  insn per cycle
                                                  #    0.09  stalled cycles per insn
         61,50,487      branches:u                       #  185.336 M/sec
            47,686      branch-misses:u                  #    0.78% of all branches

      19.125090841 seconds time elapsed

       0.006258000 seconds user
       0.027986000 seconds sys


[rishi@rishi-rogzephyrusg part_a]$ █
```

```
Request 2: Server Response: Timestamp: 1727778323
Process 1: PID: 1394, Name: (firefox), CPU Time: 6439
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Request 3: Request sent
Request 3: Server Response: Timestamp: 1727778323
Process 1: PID: 1394, Name: (firefox), CPU Time: 6444
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Request 4: Request sent
Request 4: Server Response: Timestamp: 1727778323
Process 1: PID: 1394, Name: (firefox), CPU Time: 6444
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5299

Request 5: Request sent
Request 5: Server Response: Timestamp: 1727778323
Process 1: PID: 1394, Name: (firefox), CPU Time: 6452
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Request 6: Request sent
Request 6: Server Response: Timestamp: 1727778324
Process 1: PID: 1394, Name: (firefox), CPU Time: 6453
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Request 7: Request sent
Request 7: Server Response: Timestamp: 1727778324
Process 1: PID: 1394, Name: (firefox), CPU Time: 6458
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Request 8: Request sent
Request 8: Server Response: Timestamp: 1727778324
Process 1: PID: 1394, Name: (firefox), CPU Time: 6462
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Request 9: Request sent
Request 9: Server Response: Timestamp: 1727778324
Process 1: PID: 1394, Name: (firefox), CPU Time: 6463
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300

Request 10: Request sent
Request 10: Server Response: Timestamp: 1727778324
Process 1: PID: 1394, Name: (firefox), CPU Time: 6463
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 5300


 Performance counter stats for 'taskset -c 1 ./sample_client 10':

              3.41 msec task-clock:u                     #    0.005 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               123      page-faults:u                    #   36.025 K/sec
         10,10,444      cycles:u                         #    0.296 GHz
          5,61,354      stalled-cycles-frontend:u        #   55.56% frontend cycles idle
          3,89,635      instructions:u                   #    0.39  insn per cycle
                                                  #    1.44  stalled cycles per insn
            80,790      branches:u                       #   23.662 M/sec
             8,043      branch-misses:u                  #    9.96% of all branches

       0.676095060 seconds time elapsed

       0.000000000 seconds user
       0.003439000 seconds sys


[rishi@rishi-rogzephyrusg part_a]$ ▯
```

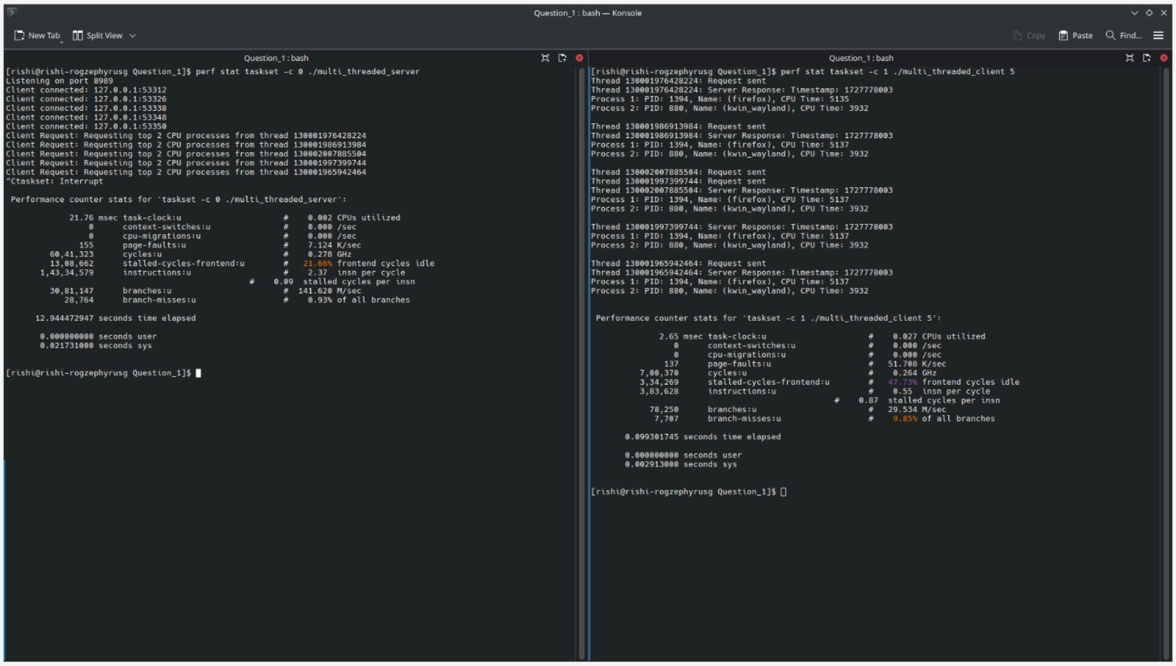
Server Analysis

Task-clock: 33.19 msec - longer processing time compared to client
Page-faults: 120 page faults - low in number - probably efficient memory access.
Cycles: 1,15,12,428 cycles - significant work on CPU as the server has to process multiple client requests, manage I/O etc.
Stalled-cycles-frontend: 21.93% - due to instruction cache misses or complex control flow - some inefficiency in fetching/decoding instructions.
Instructions per cycle (IPC): 2.48 - good efficiency of CPU
Branch misses: 0.78% is quite low - efficient branch prediction

Client Analysis

Task-clock: 3.41 msec- less time on CPU.
Page-faults: 123 indicates efficient memory handling.
Cycles: 10,10,444 - client mainly sends and receives data.
Stalled-cycles-frontend: 55.56% client waits for instruction fetching
Instructions per cycle (IPC): 0.39 - quite low as client only sends and receives data
Branch misses: 9.96% is higher than the server' - may not be very efficient

(b) Concurrent TCP client-server



```
[rishi@rishi-rogzephyrusg Question_1]$ perf stat taskset -c 0 ./multi_threaded_server
Listening on port 8989
Client connected: 127.0.0.1:53312
Client connected: 127.0.0.1:53326
Client connected: 127.0.0.1:53338
Client connected: 127.0.0.1:53348
Client connected: 127.0.0.1:53350
Client Request: Requesting top 2 CPU processes from thread 130001976428224
Client Request: Requesting top 2 CPU processes from thread 130001986913984
Client Request: Requesting top 2 CPU processes from thread 130002007885504
Client Request: Requesting top 2 CPU processes from thread 130001997399744
Client Request: Requesting top 2 CPU processes from thread 130001965942464
^Ctaskset: Interrupt

 Performance counter stats for 'taskset -c 0 ./multi_threaded_server':

             21.76 msec task-clock:u                     #    0.002 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               155      page-faults:u                    #    7.124 K/sec
         60,41,323      cycles:u                         #    0.278 GHz
         13,08,662      stalled-cycles-frontend:u        #   21.66% frontend cycles idle
       1,43,34,579      instructions:u                   #    2.37  insn per cycle
                                                  #    0.09  stalled cycles per insn
         30,81,147      branches:u                       #  141.620 M/sec
            28,764      branch-misses:u                  #    0.93% of all branches

      12.944472947 seconds time elapsed

       0.000000000 seconds user
       0.021731000 seconds sys


[rishi@rishi-rogzephyrusg Question_1]$ █
```

```
[rishi@rishi-rogzephyrusg Question_1]$ perf stat taskset -c 1 ./multi_threaded_client 5
Thread 130001976428224: Request sent
Thread 130001976428224: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5135
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001986913984: Request sent
Thread 130001986913984: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130002007885504: Request sent
Thread 130001997399744: Request sent
Thread 130002007885504: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001997399744: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932

Thread 130001965942464: Request sent
Thread 130001965942464: Server Response: Timestamp: 1727778003
Process 1: PID: 1394, Name: (firefox), CPU Time: 5137
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 3932


 Performance counter stats for 'taskset -c 1 ./multi_threaded_client 5':

              2.65 msec task-clock:u                     #    0.027 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               137      page-faults:u                    #   51.708 K/sec
          7,00,370      cycles:u                         #    0.264 GHz
          3,34,269      stalled-cycles-frontend:u        #   47.73% frontend cycles idle
          3,83,628      instructions:u                   #    0.55  insn per cycle
                                                  #    0.87  stalled cycles per insn
            78,250      branches:u                       #   29.534 M/sec
             7,707      branch-misses:u                  #    9.85% of all branches

       0.099301745 seconds time elapsed

       0.000000000 seconds user
       0.002913000 seconds sys


[rishi@rishi-rogzephyrusg Question_1]$ ▯
```

Multi-threaded Server Analysis

Task-clock: 21.76 msec indicates that the server was active for a significant duration, suggesting substantial workload handling multiple threads.
Page-faults: 155 page faults, which is still relatively low given that this is a multi-threaded server.
Cycles: 60,41,323 cycles show a higher workload compared to previous single-threaded observations, as expected in a multi-threaded setup.
Stalled-cycles-frontend: 21.66% suggests that the server efficiently fetches instructions, with fewer stalls compared to the single-threaded setup.
Instructions per cycle (IPC): 2.37 is quite efficient, indicating that the server can effectively handle multiple threads concurrently without becoming a bottleneck.
Branch misses: 0.93% is low, demonstrating efficient handling of control flow even in a multi-threaded environment.

Multi-threaded Client

Task-clock: 2.65 msec - relatively quick completion time
Page-faults: 137 - slightly higher than the single-threaded client but still within acceptable limits.
Cycles: 7,00,370 - lesser workload compared to the server; expected.
Stalled-cycles-frontend: 47.73% suggests that the client waits significantly
Instructions per cycle (IPC): 0.55 - less than server- client is not utilizing CPU efficiently.
Branch misses: 9.85% indicates that the client’s branching logic is less efficient compared to the server

Comparisons Between Multithreaded and Single-threaded

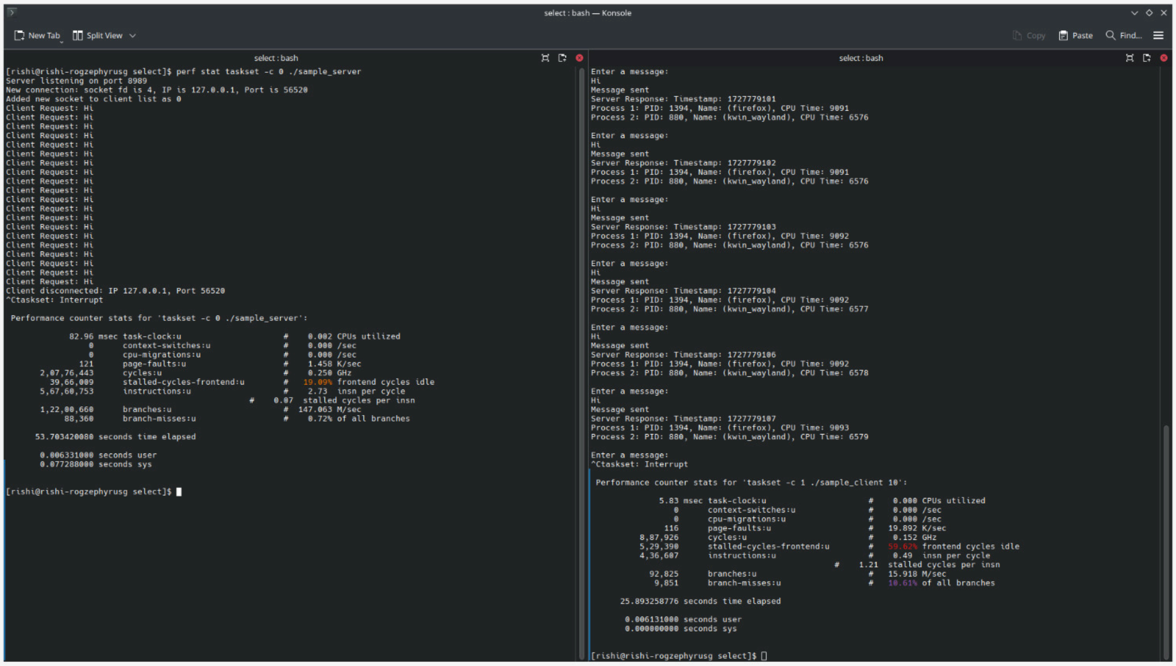
1. Server-side: The multi-threaded server handles a more significant workload efficiently, with a high IPC and low branch misprediction rate. This suggests good scalability in handling multiple threads and concurrent client requests.

   Instruction fetching is improved compared to the single-threaded setup, resulting in fewer stalled cycles.

2. Client-sider:

   The multi-threaded client doesn't perform as efficiently as the server, with a lower IPC and a high percentage of stalled cycles at the frontend, suggesting potential inefficiencies in its threading model or execution path.

(c) TCP client-server using "select"



```
[rishi@rishi-rogzephyrusg select]$ perf stat taskset -c 0 ./sample_server
Server listening on port 8989
New connection: socket fd is 4, IP is 127.0.0.1, Port is 56520
Added new socket to client list as 0
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client Request: Hi
Client disconnected: IP 127.0.0.1, Port 56520
^Ctaskset: Interrupt

 Performance counter stats for 'taskset -c 0 ./sample_server':

             82.96 msec task-clock:u                     #    0.002 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               121      page-faults:u                    #    1.458 K/sec
       2,07,76,443      cycles:u                         #    0.250 GHz
         39,66,009      stalled-cycles-frontend:u        #   19.09% frontend cycles idle
       5,67,60,753      instructions:u                   #    2.73  insn per cycle
                                                  #    0.07  stalled cycles per insn
       1,22,00,660      branches:u                       #  147.063 M/sec
            88,360      branch-misses:u                  #    0.72% of all branches

      53.703420080 seconds time elapsed

       0.006331000 seconds user
       0.077288000 seconds sys


[rishi@rishi-rogzephyrusg select]$ █
```

```
Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779101
Process 1: PID: 1394, Name: (firefox), CPU Time: 9091
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6576

Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779102
Process 1: PID: 1394, Name: (firefox), CPU Time: 9091
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6576

Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779103
Process 1: PID: 1394, Name: (firefox), CPU Time: 9092
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6576

Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779104
Process 1: PID: 1394, Name: (firefox), CPU Time: 9092
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6577

Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779106
Process 1: PID: 1394, Name: (firefox), CPU Time: 9092
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6578

Enter a message:
Hi
Message sent
Server Response: Timestamp: 1727779107
Process 1: PID: 1394, Name: (firefox), CPU Time: 9093
Process 2: PID: 880, Name: (kwin_wayland), CPU Time: 6579

Enter a message:
^Ctaskset: Interrupt

 Performance counter stats for 'taskset -c 1 ./sample_client 10':

              5.83 msec task-clock:u                     #    0.000 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
               116      page-faults:u                    #   19.892 K/sec
          8,87,926      cycles:u                         #    0.152 GHz
          5,29,390      stalled-cycles-frontend:u        #   59.62% frontend cycles idle
          4,36,607      instructions:u                   #    0.49  insn per cycle
                                                  #    1.21  stalled cycles per insn
            92,825      branches:u                       #   15.918 M/sec
             9,851      branch-misses:u                  #   10.61% of all branches

      25.893258776 seconds time elapsed

       0.006131000 seconds user
       0.000000000 seconds sys


[rishi@rishi-rogzephyrusg select]$ ▯
```

Client

Task-clock: 5.83 msec - less time spent on CPU; expected.

Page-faults: 116 is acceptable and is not of major concern.

Cycles: 8,87,926 cycles indicate that the client node has low CPU utilization.

Stalled-cycles-frontend: 59.62 % i.e ~half of the cycles are waiting, suggesting a potential inefficiency in instruction fetching.

Instructions per cycle (IPC): 0.49 indicates suboptimal CPU usage. Ideally, this should be closer to 1 for better performance.

Branch misses: 10.61% indicates inefficiencies in branch prediction

Server

Task-clock: 82.96 msec is much longer than the client - more time spent handling requests.

Cycles: 20776443 shows heavier processing.

Stalled-cycles-frontend: 19.09% is lower than the client - more efficient instruction fetching but still a significant waiting period.

Instructions per cycle (IPC): 2.73 is quite efficient use of CPU

Branch misses: 0.72% is reasonably efficient

Comparison Between Threaded and Select Server-Client Models

Server:

1. Instruction Fetching & IPC:

The multi-threaded server exhibits a lower percentage of stalled-cycles-frontend & high IPC indicating efficient instruction fetching and concurrent request handling. The select-based server has even higher IPC, suggesting it is better optimized for handling multiple requests sequentially rather than creating new threads, resulting in fewer overheads.

2. Branch Misses:

The multi-threaded server has a low branch miss rate. The select server shows an even lower branch miss rate implying better optimization for linear, non-threaded processing.

Client:

1. CPU Utilization & IPC

The multi-threaded client shows moderate CPU utilization but has a lower IPC indicating suboptimal parallel execution. The high percentage of stalled cycles suggests instruction fetching. The select-based client has an even lower IPC and high stalled-cycles-frontend indicating that it spends more time waiting for instructions. However, the select model is simpler and may avoid overheads associated with threads.

2. Branch Misses

The threaded client struggles with efficient branch prediction. The select-based client performs worse in terms of branch prediction, maybe dynamic control flow leads to more mispredictions.