Computer Network - Assignment 2

Aditya Gupta(2022031) and Aditya Jagadale(2022032)

20 August 2024

1 Question - 1

In this question , we need to write a server-client program in C . The client process and the server process should run separate VM's and communicate with each other . We are also using "taskset" to pin the processes to different CPU's . The step wise implementation is as follows along with screenshots of working .

1.1 Part 1

The first step is the beginning where the server sets up a TCP socket and listens for client connections . The code creates a TCP socket, binds it to a specific IP address and port, and listens for incoming client connections with a maximum queue length of 1000. It includes error handling at every stage (socket creation, binding, listening) to ensure that the server properly exits if any issues occur. This was the first and basic step for the server setup .

1.2 Part 2

Now we had to make a server which can handle multiple concurrent clients (multi threaded , 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) . We are creating new threads which continue to process the client connection . Also the original server socket continues to listen on the same listening port for newer incoming client connections . Now the code for the same is as follows :

- First inside the while(1) loop , we accept the new client connections in the loop using the accept() function . It waits for an incoming client connection on the listening socket (server_fd) . When a client connects , a new socket (new_socket_created) is created to handle the communication between the server and the client . The accept() call provides the client's IP address and port number in the address structure . If accept() fails , it prints an error message , closes the server socket and exits .
- We are also printing the client's information after connection using inet_ntoa(address.sin_addr) which converts the client's IP address from binary format to a human readable format. Converts the client's port number from network byte order to host byte order for readability. Also ntohs(address.sin_port) Converts the client's port number from network byte order to host byte order for readability .
- A new thread will be created for each client connection using pthread and malloc . We use the command pthread_create() to create a new thread to handle the client connection . Thus we have the defined the process of getting the top 2 processes in the custom_handler which does the rest of the work . After the response has been sent to the client connection , we close this thread using pthread_detach . This is how we are handling multiple clients .

1.3 Part 3

In this part of the question , we write the code for the client side to initiate n concurrent client connections to the TCP server , wheren n is specified as a command line argument . Each client runs in its own thread , enabling simultaneous connections with the server . We create n number of threads in this part in the client side of the code . In the for loop for the n connections , each thread calls the custom_handler() function, which handles the TCP connection . If the thread creation fails , an error message is displayed , and the program exits . After all the threads are created , another loop is used to pthread_join() each thread , ensuring that the main program waits for all the threads to complete before exiting .

1.4 Part 4

Now, in this part of the question , we need to sent a request from the client to the server to get information about the server's top TWO CPU-consuming processes . The server finds 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 and kernel mode . Now , we follow the following steps for the same :

- We have a function named get_processes for getting the top 2 processes . We scan the /proc directory in the linux environment to retrieve information about currently running processes . It populates an array of two Process structures , storing the two processes that are consuming the most CPU time. For each entry in the directory, it checks if the name is a digit (indicating a PID) and reads the corresponding /proc/[pid]/stat file to extract CPU time statistics. The total CPU time for each process is calculated, and if it exceeds the current top two processes, the array is updated accordingly. Finally, it closes the directory and prints a confirmation message. This is how we get the top two processes from this function . Now , we can call this function in the custom_handler to get the two most CPU consuming processes .
- In the custom_handler function for the threading , we store the information regarding these processes in the buffer and send this to the client connection. This way we are able to send the required information about these processes to the client .

1.5 Part 5

We already covered this in the custom_handler using the send system call to send the data to the client connection .

1.6 Part 6

Now , for this part , we make changes to the client side of the code . Now , we print the reply from the server after storing this information into the client side of the buffer . After this we close this socket and the connection between the client and the server closes for this specific port connection .

These are the main steps required to form the server-client connection as asked in the question . The code for the same is in the zip file attached in the Question-1 which contains the client code and the server code . Some screenshots for the same using 2 Virtual Machines are as follows :

CLIENT SIDE OF THE TERMINAL

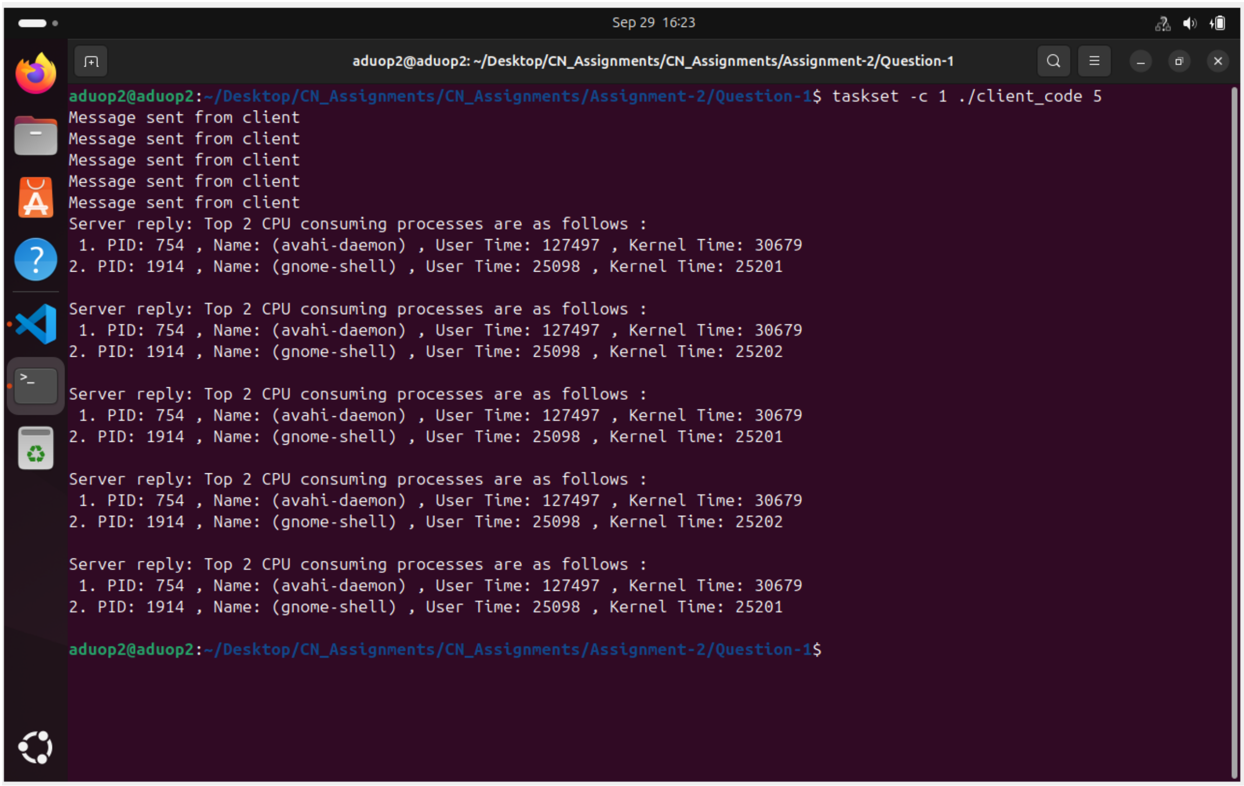

Figure 1: Client_Side

SERVER SIDE OF THE TERMINAL

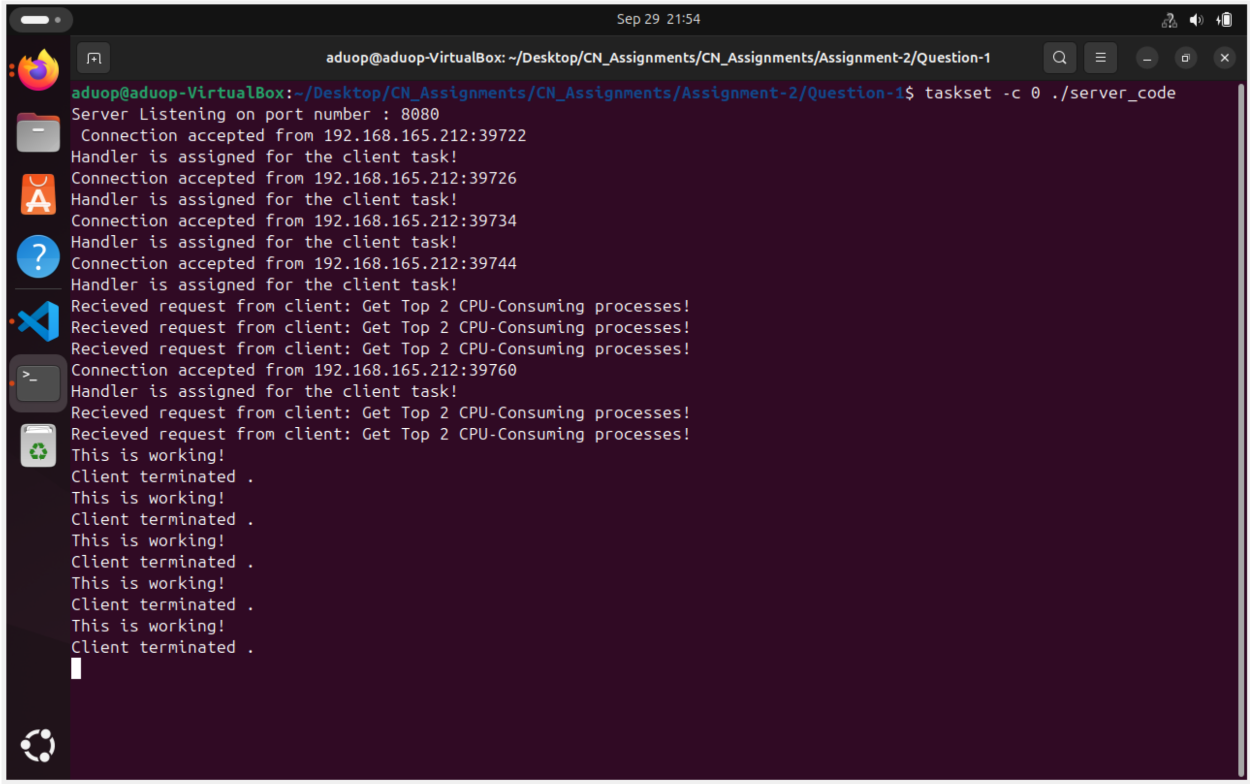

Figure 2: Server_Side

Thus we can see the messages are being sent to the client in the required quantity . This is the main task which was required to be accomplished . The top 2 tasks are also being printed .

2 Question - 2

Now, in this part of the question , we need to change the source codes from the github directory given to us according to the server code in the q1 . Now we can make the changes accordingly for this question . We need to add the functioning of getting the top 2 processes and printing them in the client side . Thus we are using the same client which we built in the question 1 to send n many number of connections(for checking purposes) and we will be changing the server side of the code i.e Sequential(Single threaded TCP client server connection) , Concurrent(Multi-Threaded TCP client server connection) and TCP client-server connection using "select" .

2.1 Part A

First , we see for the case of single-threaded TCP client-server connection . We check for two cases when the number of connections from the client are 30 and when the number of connections from the client are 100 . The results for these two cases are as follows :

- First , we check for the case of 30 Connections in single threaded TCP client-server . The results were obtained using the command **perf stat taskset -c 0 ./server** on the server side and command **perf stat taskset -c 1 ./client_code 30** . The result images are as follows :

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

          102.63 msec task-clock                       #    0.009 CPUs utilized
              11      context-switches                 #  107.179 /sec
               1      cpu-migrations                   #    9.744 /sec
             133      page-faults                      #    1.296 K/sec
    <not counted>      cpu_atom/cycles/                                                        (0.00%)
     32,34,77,258      cpu_core/cycles/                 #    3.152 GHz
    <not counted>      cpu_atom/instructions/                                                  (0.00%)
     41,68,30,285      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                      (0.00%)
      7,74,68,976      cpu_core/branches/               #  754.823 M/sec
    <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
        2,96,600      cpu_core/branch-misses/
           TopdownL1 (cpu_core)                  #     53.3 %  tma_backend_bound
                                                 #      2.4 %  tma_bad_speculation
                                                 #     21.9 %  tma_frontend_bound
                                                 #     22.4 %  tma_retiring

      12.035832585 seconds time elapsed

       0.012900000 seconds user
       0.090727000 seconds sys


iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-A$
```

Figure 3: Server_Side_Single_30

```
Performance counter stats for 'taskset -c 1 ./client_code 30':

            9.88 msec task-clock                       #    0.002 CPUs utilized
              70      context-switches                 #    7.085 K/sec
               1      cpu-migrations                   #  101.221 /sec
             196      page-faults                      #   19.839 K/sec
       50,40,736      cpu_atom/cycles/                 #    0.510 GHz                     (16.80%)
     2,01,93,099      cpu_core/cycles/                 #    2.044 GHz                     (87.09%)
       44,48,083      cpu_atom/instructions/           #    0.88  insn per cycle          (22.05%)
     1,20,14,689      cpu_core/instructions/           #    2.38  insn per cycle          (87.09%)
        8,13,824      cpu_atom/branches/               #   82.376 M/sec                   (22.05%)
       21,96,149      cpu_core/branches/               #  222.296 M/sec                   (87.09%)
          35,006      cpu_atom/branch-misses/          #    4.30% of all branches         (22.05%)
          45,946      cpu_core/branch-misses/          #    5.65% of all branches         (87.09%)
           TopdownL1 (cpu_core)                  #     23.5 %  tma_backend_bound
                                                 #      4.6 %  tma_bad_speculation
                                                 #     56.3 %  tma_frontend_bound
                                                 #     15.7 %  tma_retiring           (87.09%)
           TopdownL1 (cpu_atom)                  #     22.7 %  tma_bad_speculation
                                                 #     22.7 %  tma_retiring           (22.05%)
                                                 #     22.8 %  tma_backend_bound
                                                 #     22.8 %  tma_backend_bound_aux
                                                 #     31.7 %  tma_frontend_bound     (22.05%)

       4.349345237 seconds time elapsed

       0.001211000 seconds user
       0.008478000 seconds sys


iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-1$
```

Figure 4: Client_Side_Single_30

These are the images for the same . Now we can see from these figures that :

Table 1: Performance Analysis of Server and Client

Metric	Server	Client
Task Clock (ms)	102.63	9.88
CPU Utilization (avg)	0.009	0.002
Context Switches	11	70
Context Switch Rate (/sec)	107.179	7.085K
CPU Migrations	0	1
Page Faults	133	196
Page Fault Rate (/sec)	1.296K	19.839K
CPU Core Clock Speed (GHz)	3.152	0.510
CPU Atom Clock Speed (GHz)	-	2.044
Instructions per Cycle (Core)	-	2.38
Instructions per Cycle (Atom)	-	0.88
Branch Miss Rate (Core)	-	5.65%
Branch Miss Rate (Atom)	-	4.30%
Backend Bound (TMA)	53.3%	23.5%
Frontend Bound (TMA)	21.9%	56.3%
Retiring Instructions (TMA)	22.4%	15.7%
Bad Speculation (TMA)	2.4%	4.6%
User Time (sec)	0.0129	0.001211
System Time (sec)	0.0907	0.008478
Total Time Elapsed (sec)	12.035	4.349

Inferences

The server exhibited higher CPU utilization (0.009 vs. 0.002) and task duration (102.63 ms vs. 9.88 ms) compared to the client, indicating more resource-intensive operations. The client experienced more context switches and page faults, suggesting frequent OS interactions. While the server operated at a higher clock speed (3.152 GHz), the client achieved better instruction efficiency with an IPC of 2.38. The server was more backend-bound (53.3%) due to memory waits, while the client faced frontend-bound delays (56.3%) in fetching instructions. The server had lower bad speculation (2.4%) and retired more instructions, but its overall elapsed time (12.035 s) was longer than the client's (4.349 s).

- Now , we see for the case of 100 Connections in single threaded TCP client-server . The results are obtained using the commands **perf stat taskset -c 0 ./server** on the server side and **perf stat taskset -c 1 ./client_code 100** . The result images are as follows :

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

         297.78 msec task-clock                       #    0.002 CPUs utilized
             26      context-switches                 #   87.312 /sec
              0      cpu-migrations                   #    0.000 /sec
            134      page-faults                      #  449.990 /sec
   <not counted>      cpu_atom/cycles/                                                        (0.00%)
    82,64,37,386      cpu_core/cycles/                 #    2.775 GHz
   <not counted>      cpu_atom/instructions/                                                  (0.00%)
  1,22,84,35,797      cpu_core/instructions/
   <not counted>      cpu_atom/branches/                                                      (0.00%)
    22,83,31,699      cpu_core/branches/               #  766.769 M/sec
   <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
        8,37,313      cpu_core/branch-misses/
            TopdownL1 (cpu_core)                 #     47.5 %  tma_backend_bound
                                                 #      2.8 %  tma_bad_speculation
                                                 #     23.9 %  tma_frontend_bound
                                                 #     25.9 %  tma_retiring

   119.924163773 seconds time elapsed

     0.053342000 seconds user
     0.245575000 seconds sys


iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-A$
```

Figure 5: Server_Side_Single_100

```
Performance counter stats for 'taskset -c 1 ./client_code 100':

           30.44 msec task-clock                       #    0.000 CPUs utilized
             335      context-switches                 #   11.004 K/sec
               1      cpu-migrations                   #   32.848 /sec
             341      page-faults                      #   11.201 K/sec
   <not counted>      cpu_atom/cycles/                                                        (0.00%)
     6,11,63,145      cpu_core/cycles/                 #    2.009 GHz
   <not counted>      cpu_atom/instructions/                                                  (0.00%)
     3,48,18,554      cpu_core/instructions/
   <not counted>      cpu_atom/branches/                                                      (0.00%)
       63,11,105      cpu_core/branches/               #  207.309 M/sec
   <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
        1,23,118      cpu_core/branch-misses/
            TopdownL1 (cpu_core)                 #     20.9 %  tma_backend_bound
                                                 #      4.2 %  tma_bad_speculation
                                                 #     58.8 %  tma_frontend_bound
                                                 #     16.0 %  tma_retiring

   107.894250677 seconds time elapsed

     0.000784000 seconds user
     0.026674000 seconds sys


iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-1$
```

Figure 6: Client_Side_Single_100

Table 2: Performance Comparison: Server vs. Client for 100 Connections

Metric	**Server (taskset -c 0)**	**Client (taskset -c 1)**
Task Clock (msec)	297.78	30.44
Context Switches	26	335
CPU Migrations	0	1
Page Faults	134	341
CPU Atom/Cycles	Not counted	Not counted
CPU Core/Cycles	82,64,37,386	6,11,63,145
CPU Atom/Instructions	Not counted	Not counted
CPU Core/Instructions	1,22,84,35,797	3,48,18,554
CPU Core/Branches	22,83,31,699	63,11,105
CPU Core/Branch Misses	8,37,313	1,23,118
CPU Utilization	0.002 CPUs	0.000 CPUs
Cycles per Second (GHz)	2.775	2.009
Instructions per Second (M/sec)	766.769	207.309
TMA Backend Bound	47.5%	20.9%
TMA Bad Speculation	2.8%	4.2%
TMA Frontend Bound	23.9%	58.8%
TMA Retiring	25.9%	16.0%
Elapsed Time (seconds)	119.924	107.894

Inferences

The server processes significantly more tasks, with a higher task clock (297.78 msec) and instruction rate, compared to the client. The client experiences more context switches (335 vs. 26), indicating frequent interactions or connections. Both have low CPU utilization, but the server is more backend bound (47.5%) due to potential I/O operations, while the client faces frontend bottlenecks (58.8%). The server executes more instructions per second (766.769 M/sec vs. 207.309 M/sec), reflecting its heavier workload. Overall, the server is more I/O-constrained, while the client struggles with instruction fetching or decoding.

These were the results for the part (A) using the performed using the perf tool . Now , we move to the part (b) and check for that .

2.2 Part B

Secondly , we see for the case of multi-threaded TCP client-server connection(Concurrent TCP Client-Server connection) . We check for two cases when the number of connections from the client are 30 and when the number of connections from the client are 100 . The results for these two cases are as follows :

- First , we check for the case of 30 Connections in single threaded TCP client-server . The results were obtained using the command **perf stat taskset -c 0 ./server** on the server side and command **perf stat taskset -c 1 ./client_code 30** . The result images are as follows :

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

          54.50 msec task-clock                       #    0.011 CPUs utilized
             11      context-switches                 #  201.837 /sec
              1      cpu-migrations                   #   18.349 /sec
            255      page-faults                      #    4.679 K/sec
    <not counted>      cpu_atom/cycles/                                                        (0.00%)
     22,95,80,946      cpu_core/cycles/               #    4.213 GHz
    <not counted>      cpu_atom/instructions/                                                  (0.00%)
     42,25,23,570      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                      (0.00%)
      7,87,00,655      cpu_core/branches/             #    1.444 G/sec
    <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
         2,63,075      cpu_core/branch-misses/
          TopdownL1 (cpu_core)                 #     38.5 %  tma_backend_bound
                                               #      3.0 %  tma_bad_speculation
                                               #     27.2 %  tma_frontend_bound
                                               #     31.3 %  tma_retiring

      5.078950667 seconds time elapsed

      0.011327000 seconds user
      0.042657000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-B$
```

Figure 7: Server_Code_Multi_30

```
Performance counter stats for 'taskset -c 1 ./client_code 30':

           5.15 msec task-clock                       #    0.092 CPUs utilized
             33      context-switches                 #    6.408 K/sec
              1      cpu-migrations                   #  194.172 /sec
            194      page-faults                      #   37.669 K/sec
    <not counted>      cpu_atom/cycles/                                                        (0.00%)
      1,42,20,158      cpu_core/cycles/               #    2.761 GHz
    <not counted>      cpu_atom/instructions/                                                  (0.00%)
      1,18,90,294      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                      (0.00%)
        21,57,478      cpu_core/branches/             #  418.922 M/sec
    <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
           38,066      cpu_core/branch-misses/
          TopdownL1 (cpu_core)                 #     30.3 %  tma_backend_bound
                                               #      5.6 %  tma_bad_speculation
                                               #     41.8 %  tma_frontend_bound
                                               #     22.2 %  tma_retiring

      0.055976880 seconds time elapsed

      0.000000000 seconds user
      0.004692000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-1$
```

Figure 8: Client_Code_Multi_30

Metric	**Server (taskset -c 0)**	**Client (taskset -c 1)**
Task Clock (msec)	54.50	5.15
Context Switches	11	33
CPU Migrations	1	1
Page Faults	255	194
CPU Atom/Cycles	Not counted	Not counted
CPU Core/Cycles	22,95,80,946	1,42,20,158
CPU Atom/Instructions	Not counted	Not counted
CPU Core/Instructions	42,25,23,570	1,18,90,294
CPU Core/Branches	7,87,00,655	21,57,478
CPU Core/Branch Misses	2,63,075	38,066
CPU Utilization	0.011 CPUs	0.092 CPUs
Cycles per Second (GHz)	4.213	2.761
Instructions per Second (G/sec)	1.444	0.418
TMA Backend Bound	38.5%	30.3%
TMA Bad Speculation	3.0%	5.6%
TMA Frontend Bound	27.2%	41.8%
TMA Retiring	31.3%	22.2%
Elapsed Time (seconds)	5.079	0.0559

Inferences

The performance comparison between the server and client reveals notable differences. The server's task clock is significantly higher (54.50 msec) compared to the client (5.15 msec), indicating that the server handles a greater computational load. The server also utilizes more CPU cycles and executes more instructions per second, reflecting its more intensive workload. However, the server exhibits a higher backend bound (38.5%) compared to the client (30.3%), suggesting that more processing time is spent waiting for resources. The client, on the other hand, has a higher frontend bound (41.8%), implying bottlenecks related to instruction fetching and decoding. Additionally, the server demonstrates more efficient retiring (31.3%) compared to the client (22.2%), meaning the server completes more useful work per cycle. Overall, the server's higher CPU utilization and throughput suggest it is handling more complex operations, while the client is more constrained by frontend inefficiencies.

- Now , we see for the case of 100 Connections in single threaded TCP client-server . The results are obtained using the commands **perf stat taskset -c 0 ./server** on the server side and **perf stat taskset -c 1 ./client_code 100** . The result images are as follows :

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

          167.78 msec task-clock                       #    0.033 CPUs utilized
              56      context-switches                 #  333.763 /sec
               1      cpu-migrations                   #    5.960 /sec
             517      page-faults                      #    3.081 K/sec
    <not counted>      cpu_atom/cycles/                                                   (0.00%)
     74,63,35,368      cpu_core/cycles/                #    4.448 GHz
    <not counted>      cpu_atom/instructions/                                             (0.00%)
   1,37,55,60,128      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                 (0.00%)
     25,62,85,644      cpu_core/branches/              #    1.527 G/sec
    <not counted>      cpu_atom/branch-misses/                                            (0.00%)
         7,94,528      cpu_core/branch-misses/
           TopdownL1 (cpu_core)                 #     37.3 %  tma_backend_bound
                                                #      2.9 %  tma_bad_speculation
                                                #     27.8 %  tma_frontend_bound
                                                #     32.0 %  tma_retiring

      5.132703997 seconds time elapsed

      0.023980000 seconds user
      0.138857000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-B$
```

Figure 9: Server_Side_Multi_100

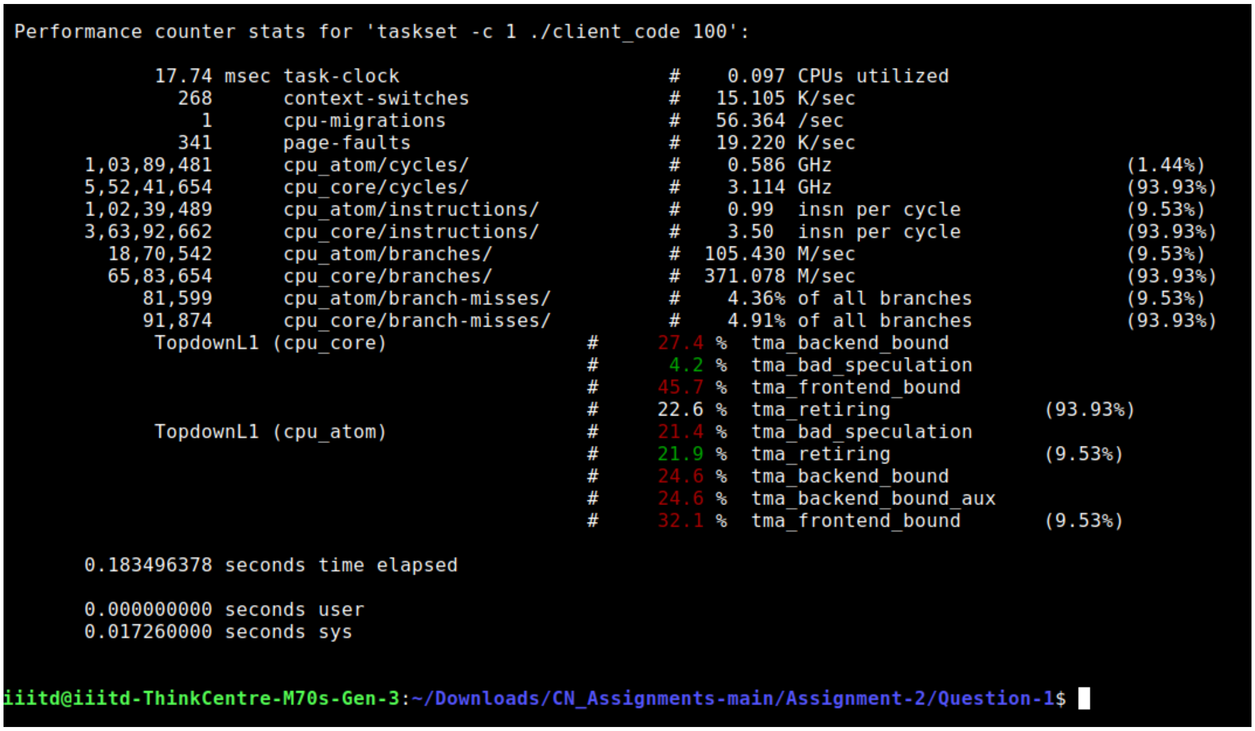

Figure 10: Client_Side_Multi_100

Metric	**Server (taskset -c 0)**	**Client (taskset -c 1)**
Task Clock (msec)	167.78	17.74
Context Switches	56	268
CPU Migrations	1	1
Page Faults	517	341
CPU Atom/Cycles	Not counted	1,03,89,481
CPU Core/Cycles	74,63,35,368	5,52,41,654
CPU Atom/Instructions	Not counted	1,02,39,489
CPU Core/Instructions	1,37,55,60,128	3,63,92,662
CPU Core/Branches	25,62,85,644	18,70,542
CPU Core/Branch Misses	7,94,528	81,599
CPU Utilization	0.033 CPUs	0.097 CPUs
Cycles per Second (GHz)	4.448	3.114
Instructions per Second (G/sec)	1.527	3.50
TMA Backend Bound	37.3%	27.4%
TMA Bad Speculation	2.9%	4.2%
TMA Frontend Bound	27.8%	45.7%
TMA Retiring	32.0%	22.6%
Elapsed Time (seconds)	5.132	0.183

Inferences

The performance comparison between the server and client highlights key distinctions. The server's task clock (167.78 msec) is much higher than the client's (17.74 msec), reflecting a heavier computational workload. The server utilizes more CPU cycles and executes a greater number of instructions per second, indicating its role in processing-intensive tasks. However, the server's higher backend bound (37.3%) compared to the client (27.4%) suggests that the server spends more time waiting for resources like memory or execution units. In contrast, the client faces more frontend bottlenecks, with a higher frontend bound (45.7%), pointing to inefficiencies in instruction fetching and decoding. The server's greater efficiency in retiring instructions (32.0%) compared to the client (22.6%) means it performs more useful work per cycle. Overall, the server handles more complex tasks efficiently, while the client is limited by frontend delays.

2.3 Part C

Lastly , we see for the case of single-threaded TCP client-server connection using select command . We check for two cases when the number of connections from the client are 30 and when the number of connections from the client are 100 . The results for these two cases are as follows :

- First , we check for the case of 30 Connections in TCP client-server connection using select command . The results were obtained using the command **perf stat taskset -c 0 ./server** on the server side and command **perf stat taskset -c 1 ./client_code 30** . The result images are as follows :

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

          99.66 msec task-clock                       #    0.001 CPUs utilized
             11      context-switches                 #  110.372 /sec
              1      cpu-migrations                   #   10.034 /sec
            133      page-faults                      #    1.334 K/sec
    <not counted>      cpu_atom/cycles/                                                       (0.00%)
     29,99,80,341      cpu_core/cycles/               #    3.010 GHz
    <not counted>      cpu_atom/instructions/                                                 (0.00%)
     42,26,31,983      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                     (0.00%)
      7,84,93,294      cpu_core/branches/             #  787.587 M/sec
    <not counted>      cpu_atom/branch-misses/                                                (0.00%)
         3,01,714      cpu_core/branch-misses/
          TopdownL1 (cpu_core)                 #     50.0 %  tma_backend_bound
                                               #      2.8 %  tma_bad_speculation
                                               #     23.0 %  tma_frontend_bound
                                               #     24.2 %  tma_retiring

    130.488953614 seconds time elapsed

      0.020306000 seconds user
      0.080212000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-C$
```

Figure 11: Server_Side_Select_30

```
Performance counter stats for 'taskset -c 1 ./client_code 30':

           6.69 msec task-clock                       #    0.002 CPUs utilized
             69      context-switches                 #   10.315 K/sec
              1      cpu-migrations                   #  149.495 /sec
            194      page-faults                      #   29.002 K/sec
    <not counted>      cpu_atom/cycles/                                                       (0.00%)
      1,65,23,513      cpu_core/cycles/               #    2.470 GHz
    <not counted>      cpu_atom/instructions/                                                 (0.00%)
      1,09,71,643      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                     (0.00%)
        20,08,918      cpu_core/branches/             #  300.323 M/sec
    <not counted>      cpu_atom/branch-misses/                                                (0.00%)
           43,134      cpu_core/branch-misses/
          TopdownL1 (cpu_core)                 #     23.6 %  tma_backend_bound
                                               #      5.4 %  tma_bad_speculation
                                               #     53.4 %  tma_frontend_bound
                                               #     17.7 %  tma_retiring

      4.397562935 seconds time elapsed

      0.000000000 seconds user
      0.005897000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-1$
```

Figure 12: Client_Code_Select_30

Metric	Server (taskset -c 0)	Client (taskset -c 1)
Task Clock (msec)	99.66	6.69
Context Switches	11	69
CPU Migrations	1	1
Page Faults	133	194
CPU Atom/Cycles	Not counted	1,65,23,513
CPU Core/Cycles	29,99,80,341	1,09,71,643
CPU Atom/Instructions	Not counted	Not counted
CPU Core/Instructions	42,26,31,983	20,98,918
CPU Core/Branches	7,84,93,294	20,08,918
CPU Core/Branch Misses	3,01,714	43,134
CPU Utilization	0.001 CPUs	0.002 CPUs
Cycles per Second (GHz)	3.010	2.470
Instructions per Second (M/sec)	787.587	300.323
TMA Backend Bound	50.0%	23.6%
TMA Bad Speculation	2.8%	5.4%
TMA Frontend Bound	23.0%	53.4%
TMA Retiring	24.2%	17.7%
Elapsed Time (seconds)	130.489	4.398
User Time (seconds)	0.020306	0.000000
System Time (seconds)	0.080212	0.005897

Inferences

The performance comparison between the server and client shows significant differences. The server's task clock (99.66 msec) is substantially higher than the client's (6.69 msec), indicating that the server handles a heavier workload. The server also consumes more CPU cycles and executes a larger number of instructions per second, reflecting its processing-intensive role. However, the client exhibits a higher frontend bound (53.4%) compared to the server's (23.0%), suggesting inefficiencies in instruction fetching for the client. Meanwhile, the server's backend bound (50.0%) is much higher than the client's (23.6%), indicating that the server spends more time waiting on resources like memory. Additionally, the client's bad speculation percentage (5.4%) is higher than the server's (2.8%), pointing to more mispredicted branches on the client side. Overall, the server handles its complex workload better, while the client faces frontend and speculation challenges.

- Now , we see for the case of 100 Connections in single threaded TCP client-server . The results are obtained using the commands **perf stat taskset -c 0 ./server** on the server side and **perf stat taskset -c 1 ./client_code 100** . The result images are as follows :

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

           238.58 msec task-clock                       #    0.002 CPUs utilized
               24      context-switches                 #  100.596 /sec
                1      cpu-migrations                   #    4.192 /sec
              134      page-faults                      #  561.661 /sec
    <not counted>      cpu_atom/cycles/                                                        (0.00%)
      82,92,82,372      cpu_core/cycles/                #    3.476 GHz
    <not counted>      cpu_atom/instructions/                                                  (0.00%)
    1,24,03,00,047      cpu_core/instructions/
    <not counted>      cpu_atom/branches/                                                      (0.00%)
      23,04,18,533      cpu_core/branches/              #  965.800 M/sec
    <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
          8,06,342      cpu_core/branch-misses/
            TopdownL1 (cpu_core)                 #     47.0 %  tma_backend_bound
                                                 #      2.8 %  tma_bad_speculation
                                                 #     23.7 %  tma_frontend_bound
                                                 #     26.5 %  tma_retiring

    134.057837889 seconds time elapsed

      0.036425000 seconds user
      0.203145000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-2/Part-C$
```

Figure 13: Server_Side_Select_100

```
 Performance counter stats for 'taskset -c 1 ./client_code 100':

            22.69 msec task-clock                       #    0.000 CPUs utilized
              331      context-switches                 #   14.587 K/sec
                3      cpu-migrations                   #  132.208 /sec
              339      page-faults                      #   14.939 K/sec
        1,70,46,547      cpu_atom/cycles/               #    0.751 GHz                        (4.94%)
        5,62,21,899      cpu_core/cycles/               #    2.478 GHz                        (98.24%)
        1,22,31,411      cpu_atom/instructions/         #    0.72  insn per cycle             (5.80%)
        3,33,13,209      cpu_core/instructions/         #    1.95  insn per cycle             (98.24%)
          22,22,838      cpu_atom/branches/             #   97.959 M/sec                      (5.80%)
          60,49,488      cpu_core/branches/             #  266.596 M/sec                      (98.24%)
           1,16,735      cpu_atom/branch-misses/        #    5.25% of all branches            (5.80%)
           1,03,365      cpu_core/branch-misses/        #    4.65% of all branches            (98.24%)
            TopdownL1 (cpu_core)                 #     18.5 %  tma_backend_bound
                                                 #      4.0 %  tma_bad_speculation
                                                 #     60.1 %  tma_frontend_bound
                                                 #     17.5 %  tma_retiring               (98.24%)
            TopdownL1 (cpu_atom)                 #     35.3 %  tma_bad_speculation
                                                 #     20.1 %  tma_retiring               (5.80%)
                                                 #     12.5 %  tma_backend_bound
                                                 #     12.5 %  tma_backend_bound_aux
                                                 #     32.1 %  tma_frontend_bound         (5.80%)

    108.818269469 seconds time elapsed

      0.002151000 seconds user
      0.017635000 seconds sys

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignments-main/Assignment-2/Question-1$
```

Figure 14: Client_Side_Select_100

Metric	Server (taskset -c 0)	Client (taskset -c 1)
Task Clock (msec)	238.58	22.69
Context Switches	24	331
CPU Migrations	1	3
Page Faults	134	339
CPU Atom/Cycles	Not counted	1,70,46,547
CPU Core/Cycles	82,92,82,372	5,62,21,899
CPU Atom/Instructions	Not counted	1,22,31,411
CPU Core/Instructions	1,24,03,00,047	3,33,13,209
CPU Core/Branches	23,04,18,533	60,49,488
CPU Core/Branch Misses	8,06,342	1,16,735
CPU Utilization	0.002 CPUs	0.000 CPUs
Cycles per Second (GHz)	3.476	2.478
Instructions per Second (M/sec)	965.800	97.959
TMA Backend Bound	47.0%	18.5%
TMA Bad Speculation	2.8%	4.0%
TMA Frontend Bound	23.7%	60.1%
TMA Retiring	26.5%	17.5%
Elapsed Time (seconds)	134.057	108.818
User Time (seconds)	0.036425	0.002151
System Time (seconds)	0.203145	0.017635

Inferences

The analysis of the performance metrics between the server and client shows clear differences in task processing and CPU behavior. The server's task clock (238.58 msec) is significantly higher than the client's (22.69 msec), reflecting a heavier load on the server. However, the client has far more context switches (331) and page faults (339), indicating frequent multitasking and memory management challenges. In terms of CPU instructions and cycles, the server again outperforms the client with 82.9 billion cycles and 124 billion instructions, compared to the client's 5.62 billion cycles and 3.33 billion instructions. Both systems exhibit minimal CPU utilization, with the server utilizing 0.002 CPUs and the client at 0.000 CPUs, suggesting low overall resource use. The server also shows a higher backend-bound percentage (47.0%) than the client (18.5%), indicating more time spent waiting on resources like memory or I/O.

2.4 Analysis based on these different Approaches

Now we can analyze based on the factors we get in the above approaches for different algorithms . We are going to compare for 100 connections from the client and give our results :

Table 3: Comparison of Task Clock, Context Switches, and CPU Utilization for Different Approaches

Metric	**Single-Threaded TCP (100 Conn)**	**Multi-Threaded TCP (100 Conn)**	**TCP Select-Based Connection (100 Conn)**
Task Clock (msec)	297.78	167.78	238.58
Context Switches	26	56	24
CPU Utilization	0.002 CPUs	0.033 CPUs	0.002 CPUs

Thus , we infer the following from these :

- **Task Clock**

- * **Single-threaded TCP connection**: Highest task clock (297.78 msec), indicating longer time per task.
  * **Multi-threaded TCP connection**: Lowest task clock (167.78 msec), reflecting faster task completion due to parallelism.
  * **Select-based TCP connection**: Moderate task clock (238.58 msec), suggesting balanced performance.

- **Context Switches**
  * **Single-threaded TCP connection**: Minimal context switches (26), showing reduced multitasking overhead.
  * **Multi-threaded TCP connection**: Highest context switches (56), indicating frequent task switching due to threading.
  * **Select-based TCP connection**: Slightly lower context switches (24), highlighting efficient management with minimal task switching.

- **CPU Utilization**
  * **Single-threaded TCP connection**: Low CPU utilization (0.002 CPUs), meaning it uses fewer CPU resources.
  * **Multi-threaded TCP connection**: Highest CPU utilization (0.033 CPUs), demonstrating the use of more processing power due to multi-threading.
  * **Select-based TCP connection**: Similar to single-threaded (0.002 CPUs), showing low CPU demand despite handling tasks effectively.