Vipul Verma 2022577
Wasif Ali 2022583

Report for Client-Server Application (Single Threading, Multi-threading, and System-level Threading)

Overview:

This report discusses three server implementations (single-threaded, multi-threaded, and system-level threaded using **select()** as well as a client program. All server implementations handle requests from clients to retrieve the top two CPU-consuming processes on the server machine using **/proc/[pid]/stat** data. The client sends a request to the server , and the server responds with the requested information. These implementations vary in their approach to handling multiple clients.

Client Implementation

The client program is designed to connect to the server on a predefined port, specifically port 8081. It utilizes various libraries essential for network programming and threading. The key libraries include:

`arpa/inet.h`: Facilitates network programming using IPv4.

`unistd.h`: Provides access to standard Unix functions.

`pthread.h`: Enables the use of POSIX threads for creating multithreaded clients.

Functionality:

Upon execution, the client establishes a connection to the server and sends a request string, such as **`"GET_CPU_INFO"`**, to retrieve the top two CPU-consuming processes. The client then waits for a response from the server and prints the received data to the console. For multi-threaded clients, the implementation allows for multiple threads to simulate concurrent client connections.

Concurrency Mechanism: Multiple threads can be created to simulate several clients making concurrent requests to the server.

Single-threaded Server

stdio.h, **stdlib.h**, **string.h**: Standard input-output and string handling

dirent.h: Access to the **/proc** directory

arpa/inet.h: Network programming (IPv4)

unistd.h: Unix standard functions

Functionality: This server listens for incoming client connections on port 8081 and processes requests sequentially. Upon accepting a client connection, the server reads the request, retrieves the top two CPU-consuming processes from the `/proc` file system, and sends the data back to the client. After handling the request, the server closes the connection and awaits the next client.

Key Functions: **read_process_stat():** Reads the **/proc/[pid]/stat** file to extract process details such as **pid**, **command**, **user time**, **and kernel time.**

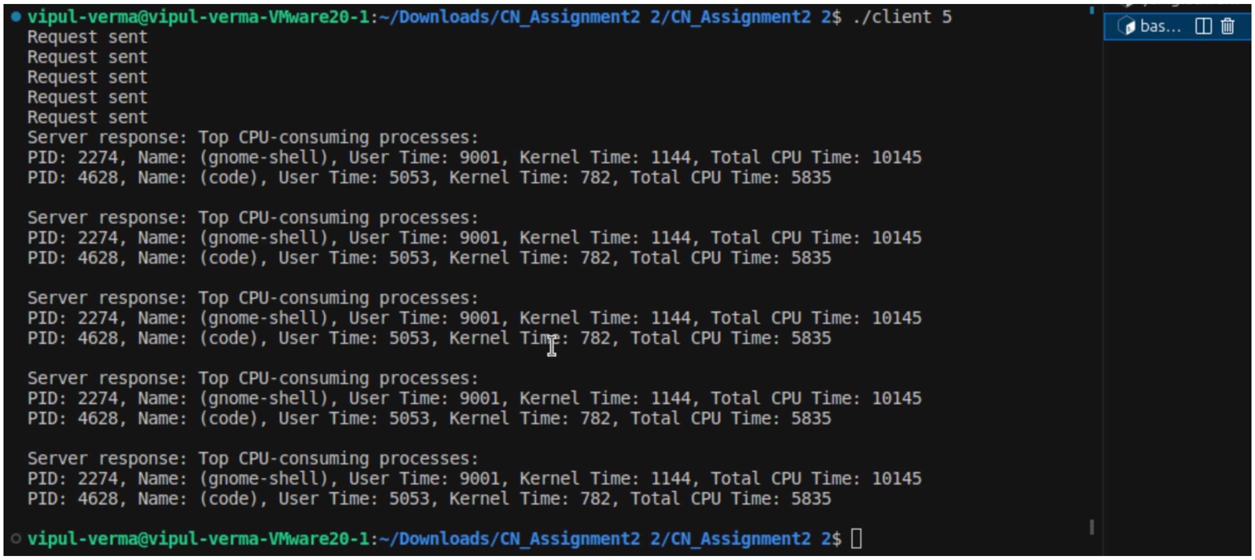
get_top_cpu_processes(): Iterates through the **/proc** directory, reads process stats, and sorts the processes by CPU usage to identify the top two.

Output :

Question 1: Single Threading (Server side)

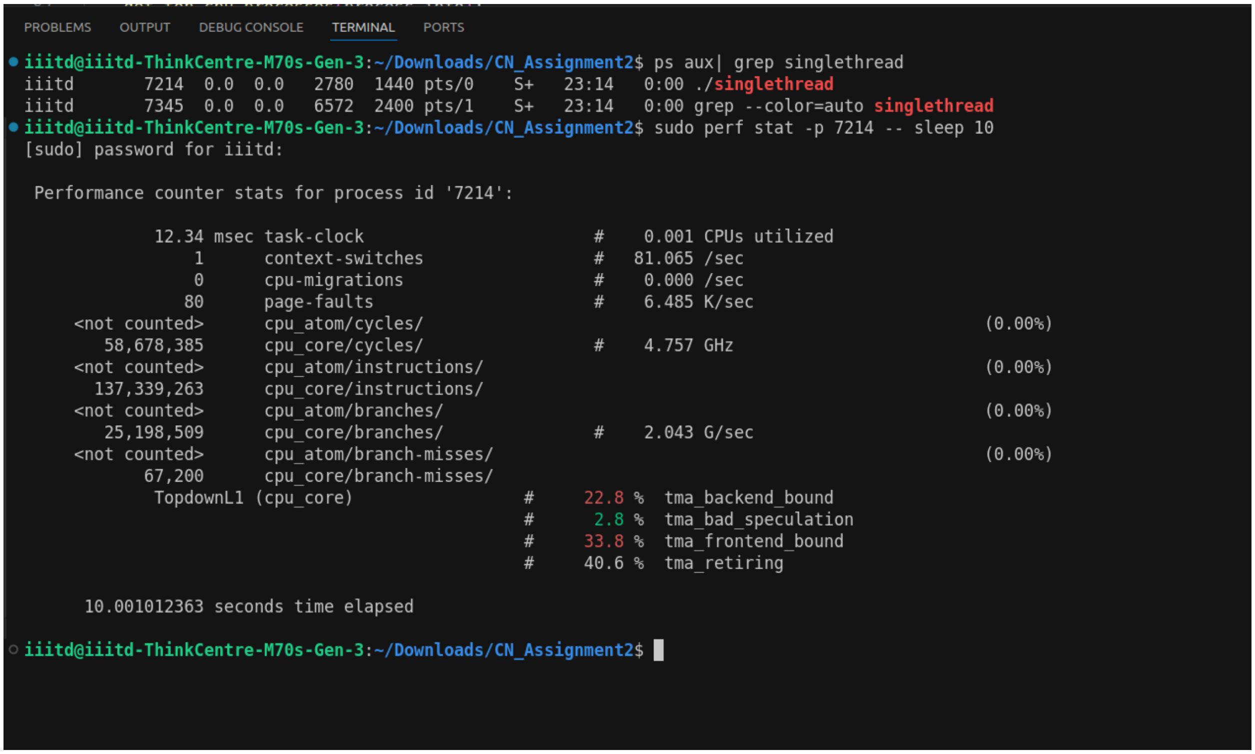
```
vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ make singlethread
gcc -Wall -Werror -pthread -o singlethread SingleThreading_Server.c
vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ ./singlethread
Server is listening on port 8081
Connection accepted from client IP: 192.168.246.128, Port: 34254
Received:
Connection accepted from client IP: 192.168.246.128, Port: 34262
Received:
Connection accepted from client IP: 192.168.246.128, Port: 34278
Received:
Connection accepted from client IP: 192.168.246.128, Port: 34266
Received:
Connection accepted from client IP: 192.168.246.128, Port: 34292
Received:
```

Single Threading (Client Side) :



```
vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ ./client 5
Request sent
Request sent
Request sent
Request sent
Request sent
Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9001, Kernel Time: 1144, Total CPU Time: 10145
PID: 4628, Name: (code), User Time: 5053, Kernel Time: 782, Total CPU Time: 5835

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9001, Kernel Time: 1144, Total CPU Time: 10145
PID: 4628, Name: (code), User Time: 5053, Kernel Time: 782, Total CPU Time: 5835

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9001, Kernel Time: 1144, Total CPU Time: 10145
PID: 4628, Name: (code), User Time: 5053, Kernel Time: 782, Total CPU Time: 5835

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9001, Kernel Time: 1144, Total CPU Time: 10145
PID: 4628, Name: (code), User Time: 5053, Kernel Time: 782, Total CPU Time: 5835

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9001, Kernel Time: 1144, Total CPU Time: 10145
PID: 4628, Name: (code), User Time: 5053, Kernel Time: 782, Total CPU Time: 5835

vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$
```

Question 2: For Single Threading



```
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ ps aux| grep singlethread
iiitd       7214  0.0  0.0   2780  1440 pts/0    S+   23:14   0:00 ./singlethread
iiitd       7345  0.0  0.0   6572  2400 pts/1    S+   23:14   0:00 grep --color=auto singlethread
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ sudo perf stat -p 7214 -- sleep 10
[sudo] password for iiitd:

 Performance counter stats for process id '7214':

             12.34 msec task-clock                       #    0.001 CPUs utilized
                 1      context-switches                 #   81.065 /sec
                 0      cpu-migrations                   #    0.000 /sec
                80      page-faults                      #    6.485 K/sec
     <not counted>      cpu_atom/cycles/                                                        (0.00%)
        58,678,385      cpu_core/cycles/                 #    4.757 GHz
     <not counted>      cpu_atom/instructions/                                                  (0.00%)
       137,339,263      cpu_core/instructions/
     <not counted>      cpu_atom/branches/                                                      (0.00%)
        25,198,509      cpu_core/branches/               #    2.043 G/sec
     <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
            67,200      cpu_core/branch-misses/
             TopdownL1 (cpu_core)                 #     22.8 %  tma_backend_bound
                                                  #      2.8 %  tma_bad_speculation
                                                  #     33.8 %  tma_frontend_bound
                                                  #     40.6 %  tma_retiring

      10.001012363 seconds time elapsed

iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$
```

Multi-threaded Server: Same as single-threaded server, with the addition of:

pthread.h: POSIX thread library for multi-threading.

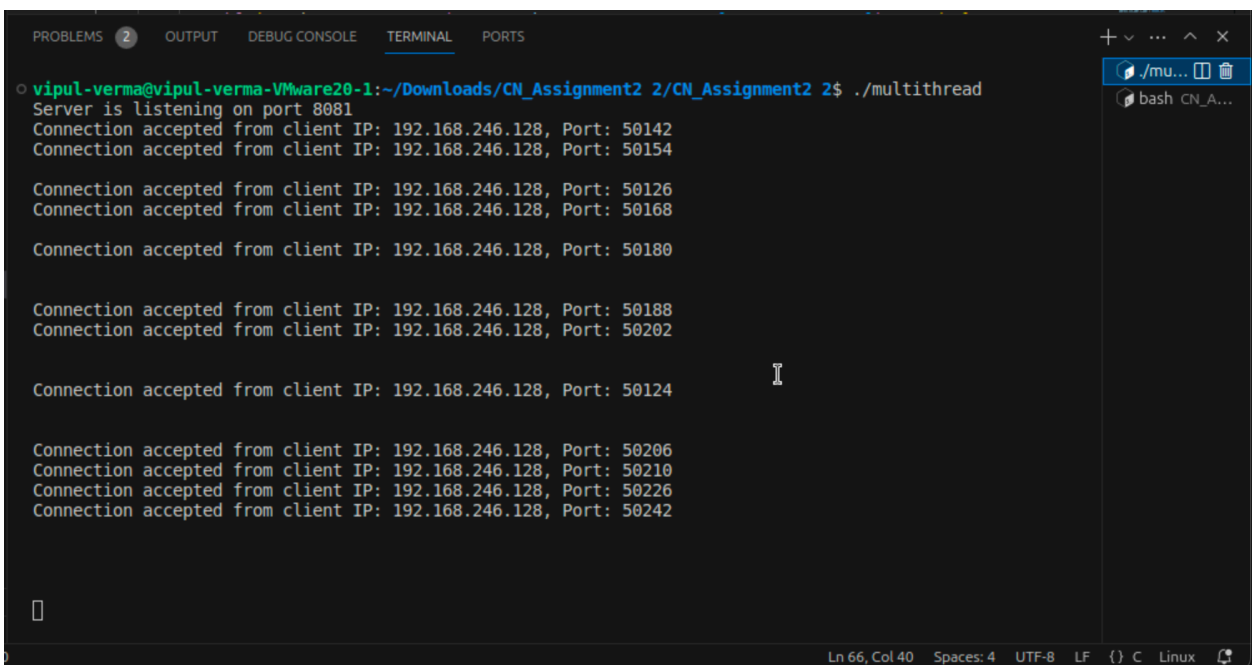
Functionality: This server spawns a new thread for each client connection. When a client connects, a new thread is created using `pthread_create()`, which then handles the client's request independently. This thread processes the request in the same manner as the single-threaded server, retrieving the top CPU-consuming processes.

Key Functions: **handle_client():** This function is executed by each thread to handle a client's request.

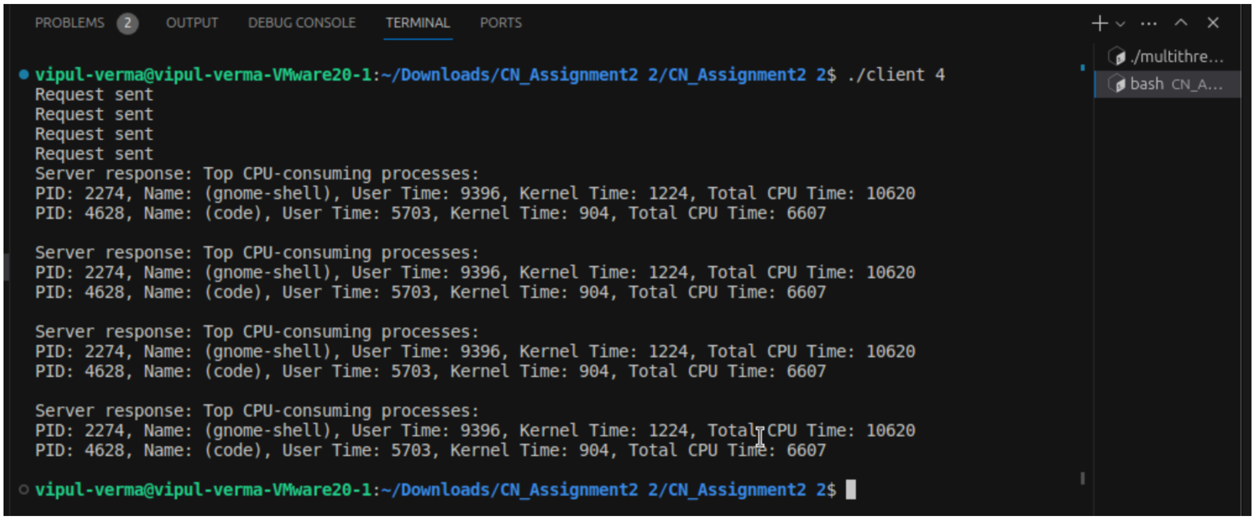
Concurrency Mechanism: The server allows concurrent connections by creating threads dynamically. Each client request is handled independently, improving scalability over the single-threaded approach.

Output:

Question 1: Multi Threading (Server side)

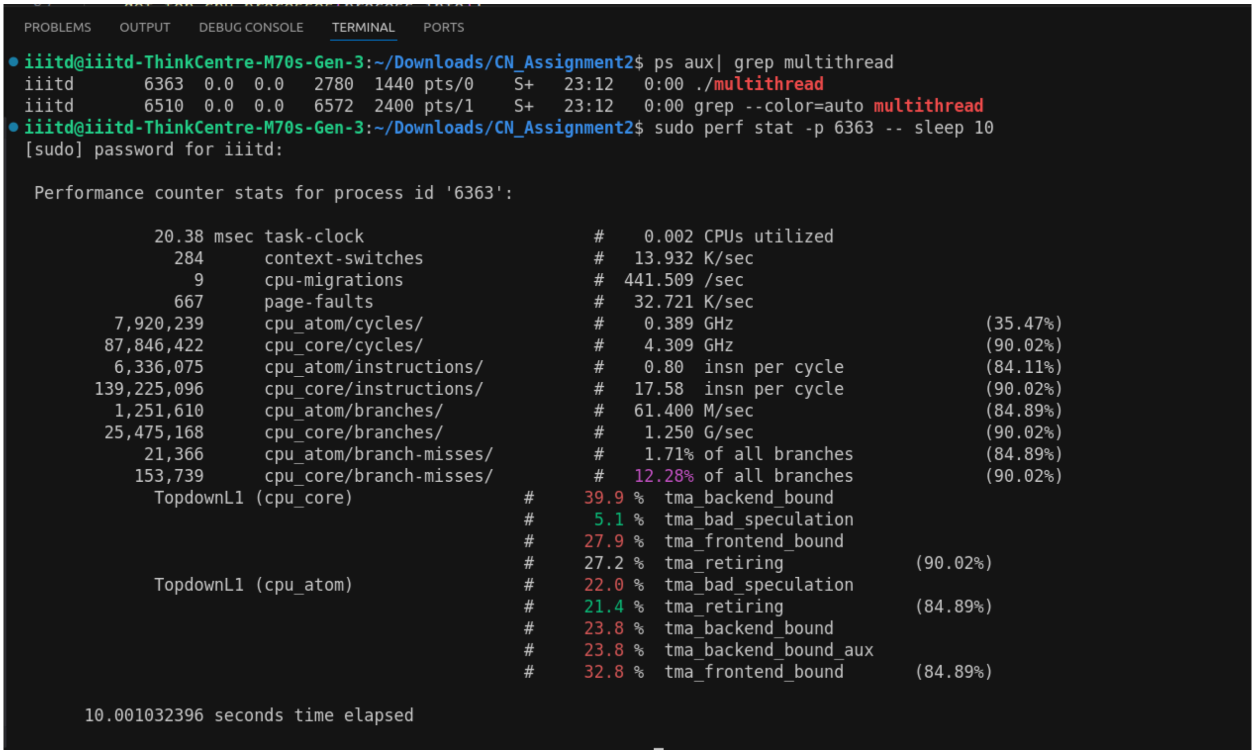


Question 1: Multi Threading (Client side)



```
vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ ./client 4
Request sent
Request sent
Request sent
Request sent
Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9396, Kernel Time: 1224, Total CPU Time: 10620
PID: 4628, Name: (code), User Time: 5703, Kernel Time: 904, Total CPU Time: 6607

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9396, Kernel Time: 1224, Total CPU Time: 10620
PID: 4628, Name: (code), User Time: 5703, Kernel Time: 904, Total CPU Time: 6607

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9396, Kernel Time: 1224, Total CPU Time: 10620
PID: 4628, Name: (code), User Time: 5703, Kernel Time: 904, Total CPU Time: 6607

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9396, Kernel Time: 1224, Total CPU Time: 10620
PID: 4628, Name: (code), User Time: 5703, Kernel Time: 904, Total CPU Time: 6607

vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ 
```

Question 2: Multi Threading



```
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ ps aux| grep multithread
iiitd       6363  0.0  0.0   2780  1440 pts/0    S+   23:12   0:00 ./multithread
iiitd       6510  0.0  0.0   6572  2400 pts/1    S+   23:12   0:00 grep --color=auto multithread
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ sudo perf stat -p 6363 -- sleep 10
[sudo] password for iiitd:

 Performance counter stats for process id '6363':

             20.38 msec task-clock                       #    0.002 CPUs utilized
               284      context-switches                 #   13.932 K/sec
                 9      cpu-migrations                   #  441.509 /sec
               667      page-faults                      #   32.721 K/sec
         7,920,239      cpu_atom/cycles/                 #    0.389 GHz                         (35.47%)
        87,846,422      cpu_core/cycles/                 #    4.309 GHz                         (90.02%)
         6,336,075      cpu_atom/instructions/           #    0.80  insn per cycle              (84.11%)
       139,225,096      cpu_core/instructions/           #   17.58  insn per cycle              (90.02%)
         1,251,610      cpu_atom/branches/               #   61.400 M/sec                       (84.89%)
        25,475,168      cpu_core/branches/               #    1.250 G/sec                       (90.02%)
            21,366      cpu_atom/branch-misses/          #    1.71% of all branches             (84.89%)
           153,739      cpu_core/branch-misses/          #   12.28% of all branches             (90.02%)
             TopdownL1 (cpu_core)                 #     39.9 %  tma_backend_bound
                                                  #      5.1 %  tma_bad_speculation
                                                  #     27.9 %  tma_frontend_bound
                                                  #     27.2 %  tma_retiring             (90.02%)
             TopdownL1 (cpu_atom)                 #     22.0 %  tma_bad_speculation
                                                  #     21.4 %  tma_retiring             (84.89%)
                                                  #     23.8 %  tma_backend_bound
                                                  #     23.8 %  tma_backend_bound_aux
                                                  #     32.8 %  tma_frontend_bound       (84.89%)

      10.001032396 seconds time elapsed
```

System-level Threading Server (Using select()): Same as multi-threaded server, with the addition of **sys/select.h**: Allows monitoring of multiple file descriptors (sockets) using **select()** system call.

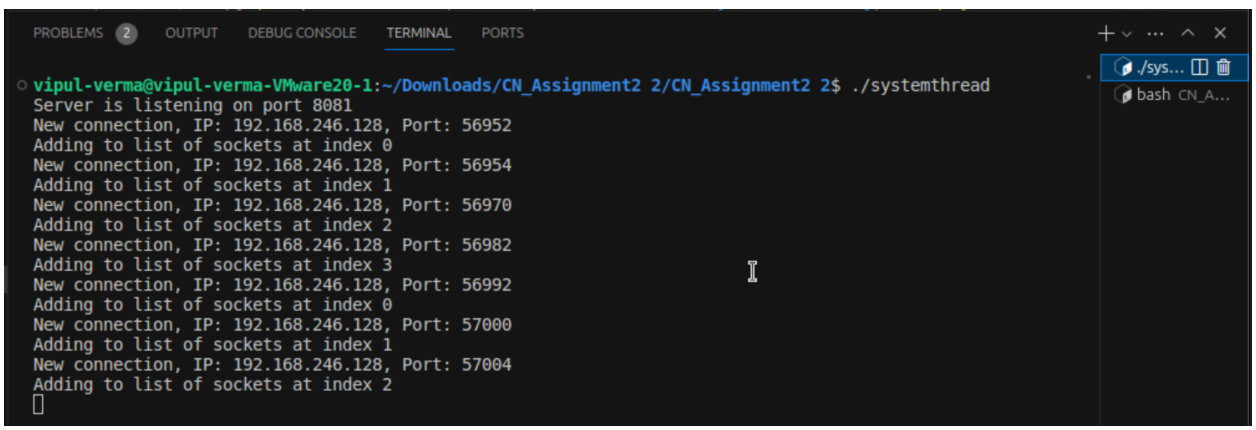
Functionality: Instead of spawning new threads for each client, this server maintains an array of client sockets and uses `select()` to monitor these sockets, as well as the main server socket. When a new client connects, the server accepts the connection and adds the client socket to the set monitored by `select()`. The server then processes any incoming requests from clients as they are received.

Concurrency Mechanism: This server can handle multiple clients simultaneously without spawning new threads. Instead, it uses a single main loop and monitors multiple clients using **select(), e**nsuring that no client blocks others

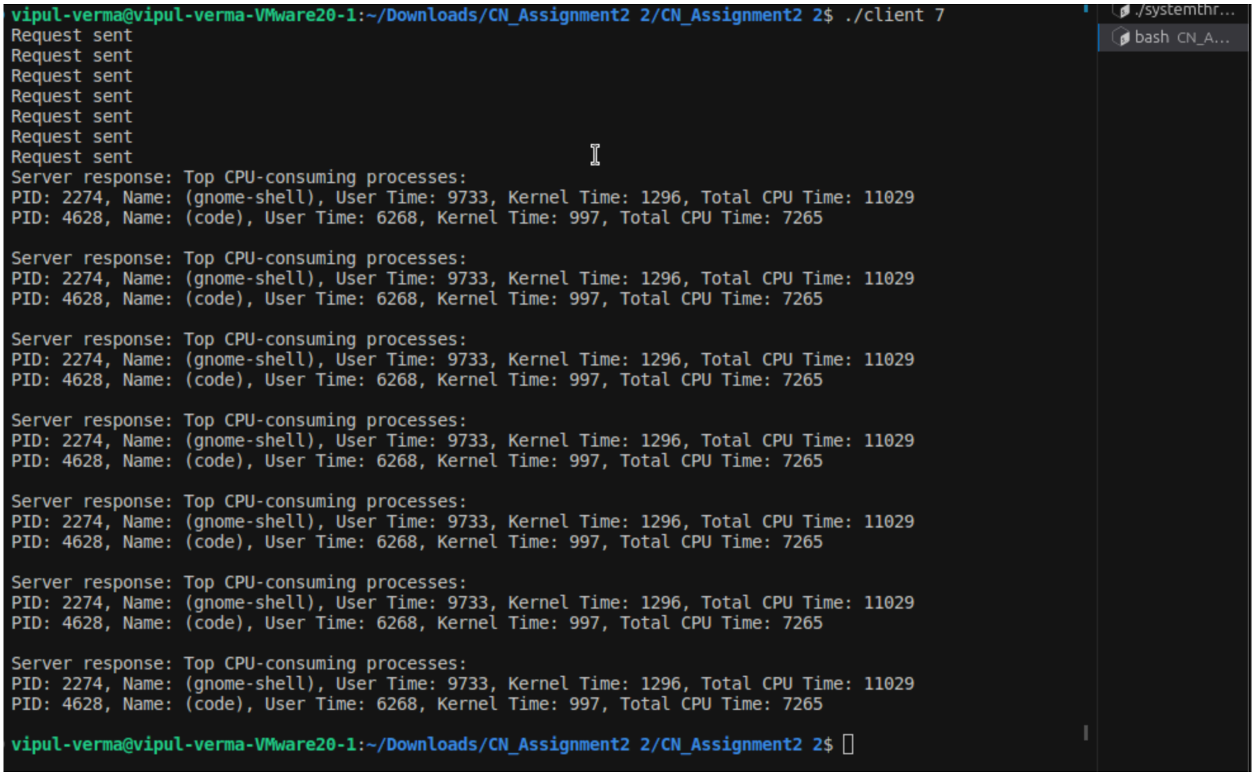
Key Features: Server does not need to create new threads for each client, making it more efficient in terms of memory and CPU usage. Handles up to **MAX_CLIENTS** simultaneously, making it suitable for systems where high concurrency is required.Monitors multiple sockets for reading and writing, ensuring that no client starves.

Output:

Question 1: System-level Threading (Server side)

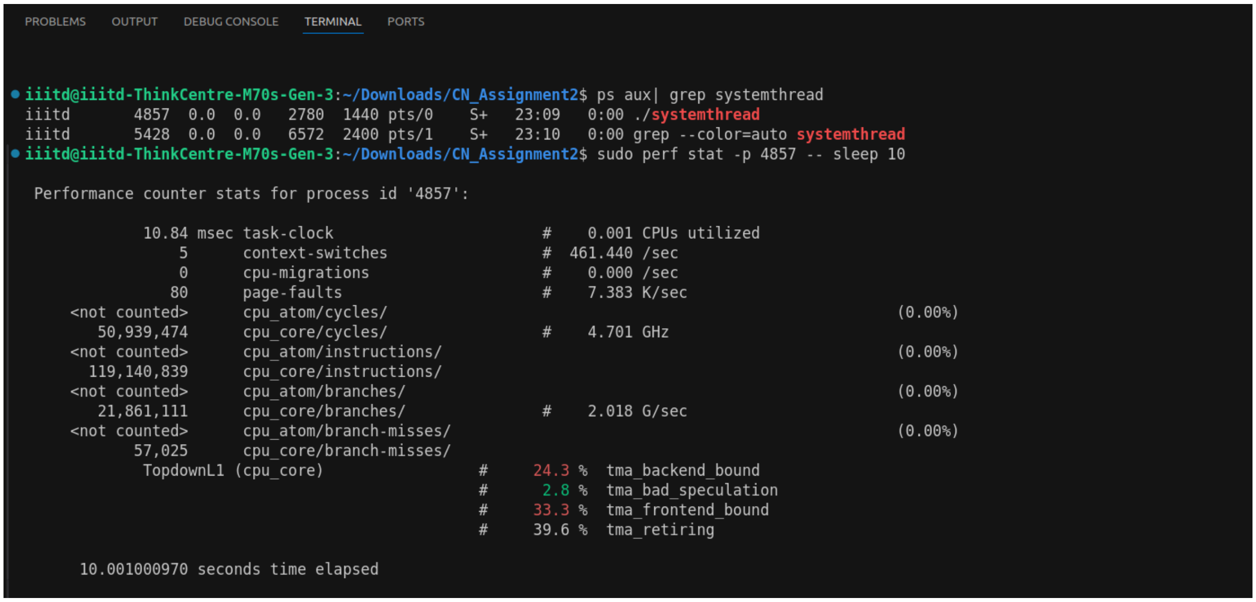


Question 1: System-level Threading (Client side)



```
vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ ./client 7
Request sent
Request sent
Request sent
Request sent
Request sent
Request sent
Request sent
Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

Server response: Top CPU-consuming processes:
PID: 2274, Name: (gnome-shell), User Time: 9733, Kernel Time: 1296, Total CPU Time: 11029
PID: 4628, Name: (code), User Time: 6268, Kernel Time: 997, Total CPU Time: 7265

vipul-verma@vipul-verma-VMware20-1:~/Downloads/CN_Assignment2 2/CN_Assignment2 2$ 
```

Question 2: System-level Threading



```
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ ps aux| grep systemthread
iiitd       4857  0.0  0.0   2780  1440 pts/0    S+   23:09   0:00 ./systemthread
iiitd       5428  0.0  0.0   6572  2400 pts/1    S+   23:10   0:00 grep --color=auto systemthread
iiitd@iiitd-ThinkCentre-M70s-Gen-3:~/Downloads/CN_Assignment2$ sudo perf stat -p 4857 -- sleep 10

 Performance counter stats for process id '4857':

             10.84 msec task-clock                       #    0.001 CPUs utilized
                 5      context-switches                 #  461.440 /sec
                 0      cpu-migrations                   #    0.000 /sec
                80      page-faults                      #    7.383 K/sec
     <not counted>      cpu_atom/cycles/                                                        (0.00%)
        50,939,474      cpu_core/cycles/                 #    4.701 GHz
     <not counted>      cpu_atom/instructions/                                                  (0.00%)
       119,140,839      cpu_core/instructions/
     <not counted>      cpu_atom/branches/                                                      (0.00%)
        21,861,111      cpu_core/branches/               #    2.018 G/sec
     <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
            57,025      cpu_core/branch-misses/
             TopdownL1 (cpu_core)                 #     24.3 %  tma_backend_bound
                                                  #      2.8 %  tma_bad_speculation
                                                  #     33.3 %  tma_frontend_bound
                                                  #     39.6 %  tma_retiring

      10.001000970 seconds time elapsed
```

Why is Context Switch different in Single ,Multi, System-level Threadings ?

In a single-threaded program, there's only one task running at a time, so the CPU doesn't need to switch between tasks often, leading to very few context switches, like 1 per second. In a multithreaded program using user-level threads (managed by the program itself), the CPU switches between tasks more frequently, causing a higher number of context switches, like 284 per second. When using system-level threads (managed by the operating system), the switching is more efficient, leading to fewer context switches, like 5 per second, because the OS handles the scheduling better.