CN Assignment-2 (Socket Programming and Perf Tool)

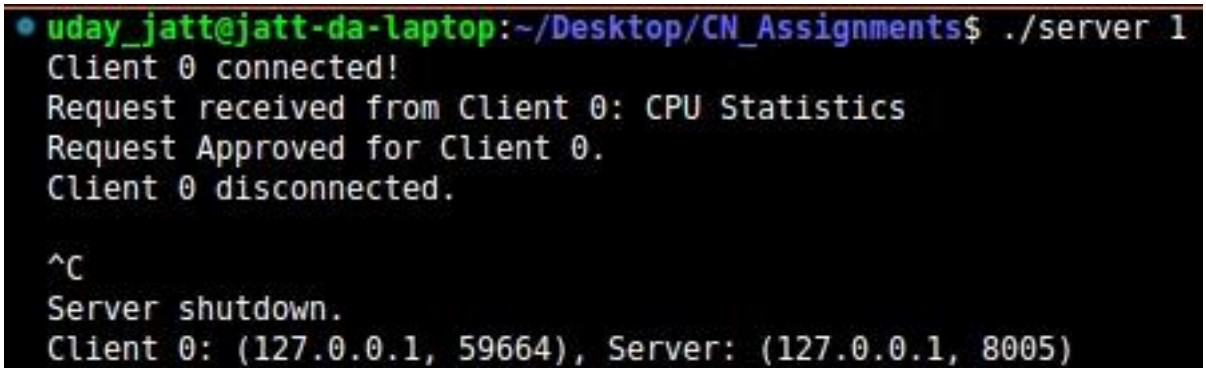
Name: Uday Kumar Sangwan
Roll No.: 2022547

Name: Utkarsh Dhilliwal
Roll No.: 2022551

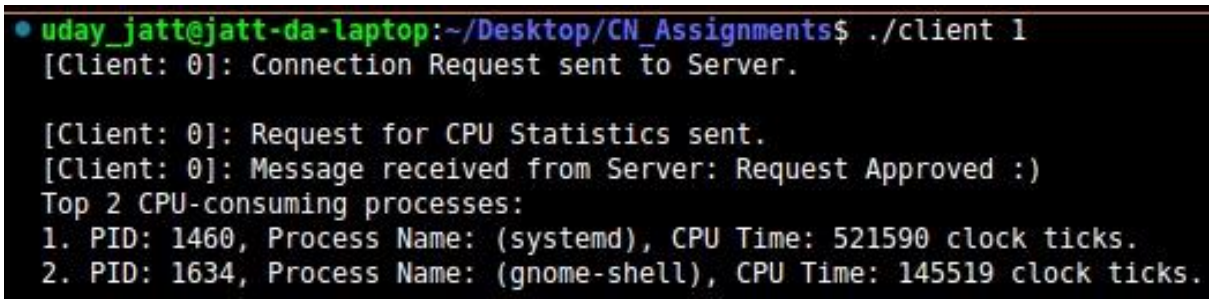
Question 1:
Multithreaded Socket Programming: passing number of clients to establish connection with server [n connection].

Outputs:
Single Client connection to Server:

Server Output- It displays which all clients have connected, the request made by clients and then the server approves and sends a confirmation to the client. Upon the request being approved CPU Statistics are sent to the client for the top 2 process (decided based on CPU usage [user time + kernel time]. Once the statistics are received I am disconnecting the client and printing the IP and Port Address of both the server and client upon ending server (Ctrl + C). (IP, Port No.) {Format}
The screenshot for the same is added below.

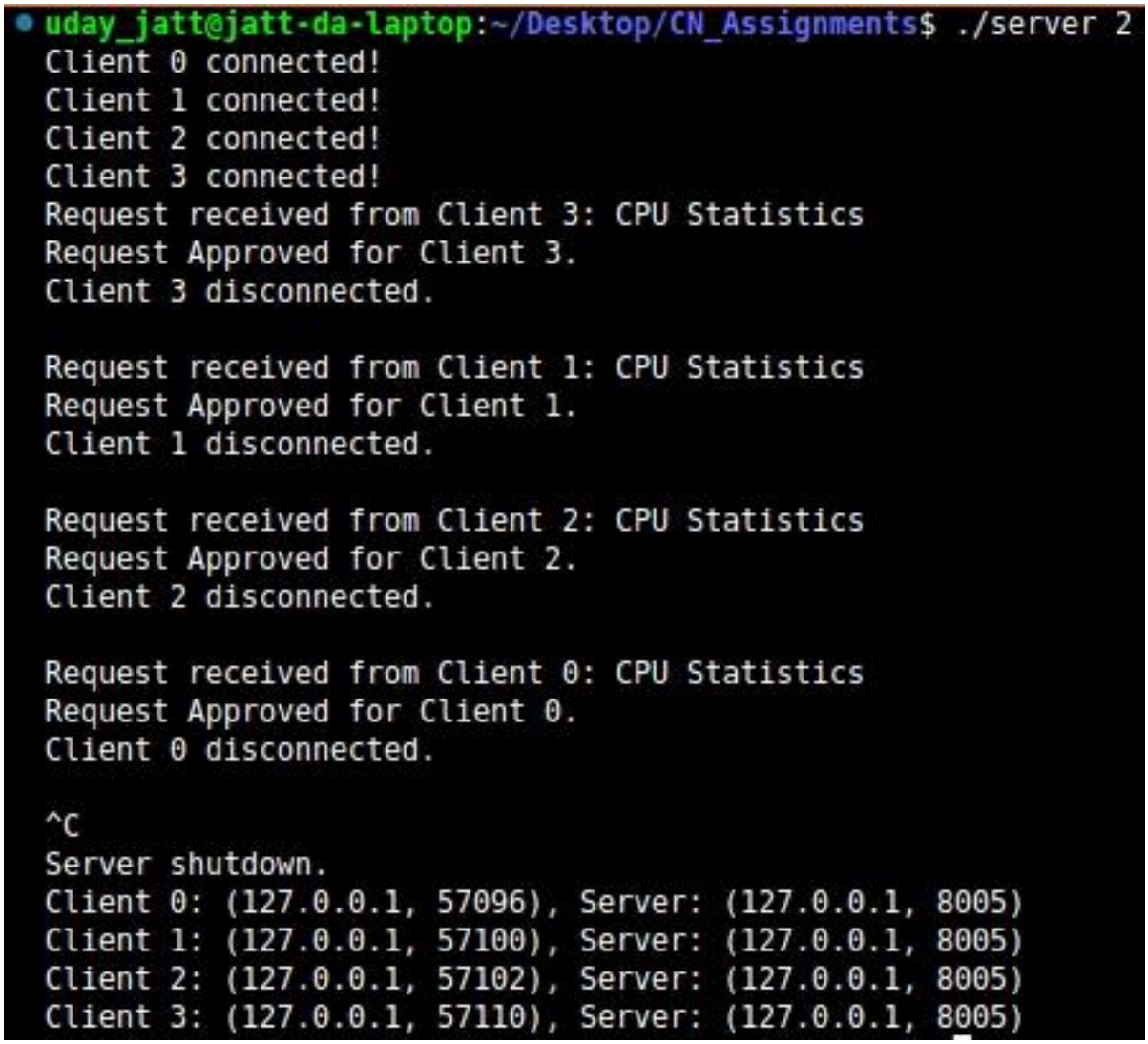


Client Output- Client makes a connection request and once its connection is established it sends a request for CPU Statistics. Once the request is approved it receives the statistics from the server; we print (PID, Process name and CPU Time = user time + kernel time). In this case we see the top 2 processes as systemd (array of system components for Linux) and gnome-shell (graphical shell).
The screenshot for the same is added below.

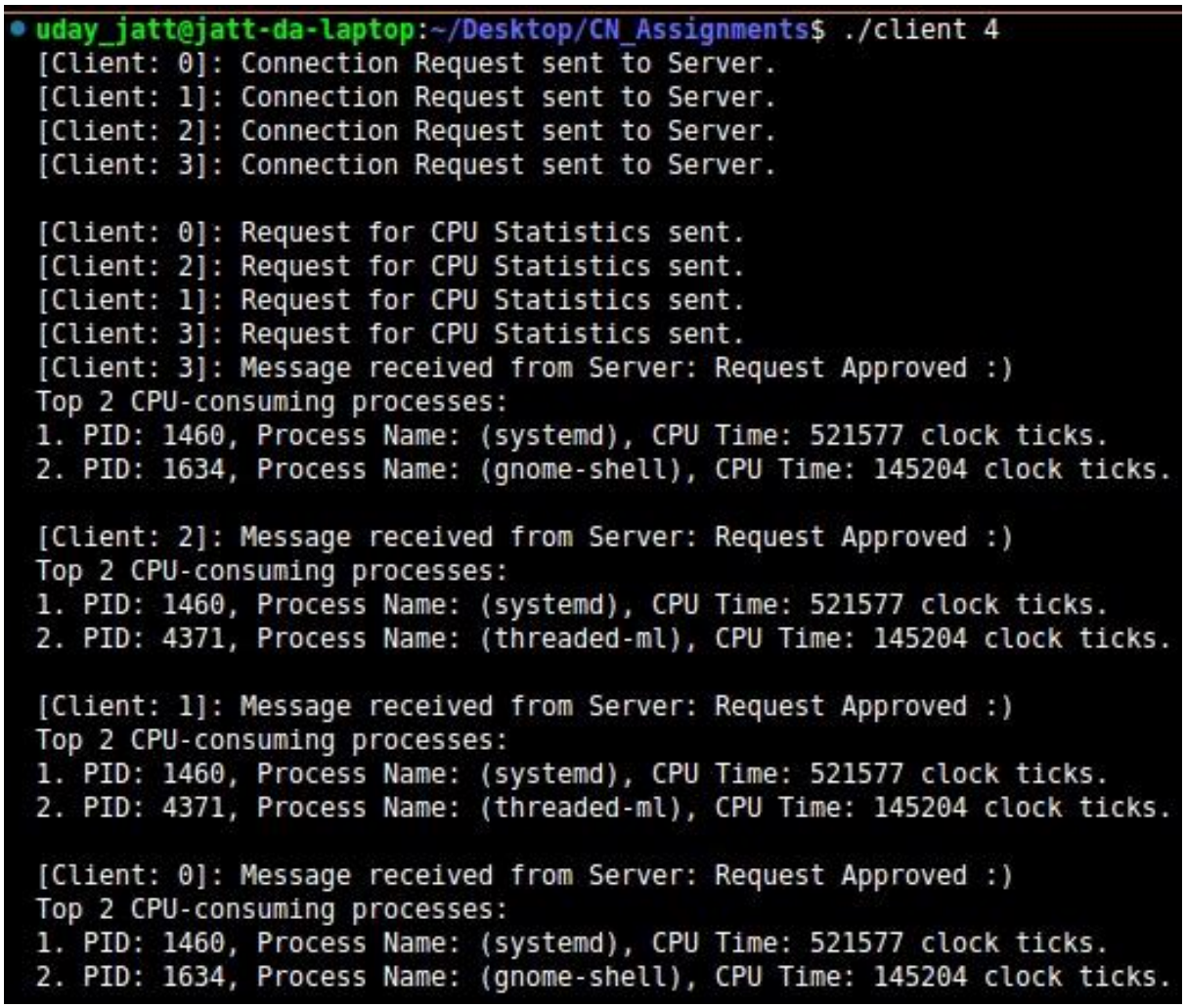


Multiple Client Connections to Server:

Server Output- It displays which all clients have connected, the request made by clients and then the server approves and sends a confirmation to the client. Upon the request being approved CPU Statistics are sent to the client for the top 2 process (decided based on CPU usage [user time + kernel time]. Once the statistics are received I am disconnecting the client and printing the IP and Port Address of both the server and clients upon ending server (Ctrl + C). (IP, Port No.) {Format}. The only difference is we specify at a time how many clients can concurrently run on the server by passing it as an argument, hence 'n' number of clients can connect to the server at once and make requests. [In this case 2]
The screenshot for the same is added below.



Client Output- Clients make their connection requests to the server and once its connection is established it sends a request for CPU Statistics. Once the request is approved for a client it receives the statistics from the server; we print (PID, Process name and CPU Time = user time + kernel time). The Client which receives the approval first can be any of them since we are not specifying any sequential mechanism. Hence the reason we see Client 1 getting approval before Client 0 or the other clients as well. Again, we can specify 'n' number of client connections that are sent to the server which will be concurrently connected to the server.
The screenshot for the same is added below.



For reading the statistics, like mentioned in the hint we used the open sys-call to open the proc directory which contains details of all the processes running in the OS with their corresponding details. We parse the PID's in the proc file for its respective details [proc/pid/stat]. Since we are to only find the process Id, name, u-time and s-time, we iterate over the field entries, retrieve these required details and store it in a buffer to send it to the client.
With this all parts of Question 1 are covered, mainly concurrent multithreaded TCP Connection, returning 4 tuple IP, Port addresses [Client & Server] and Top 2 CPU processes.

Question 2:
Using perf tool for retrieving CPU performance metrics. We will be running it only on one core by specifying taskset -c 1.

a) **Single Threaded TCP Client-Server:**

Server Output & Performance:

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 0 ./server 1
Client 0 connected!
Client 1 connected!
Client 2 connected!
Client 3 connected!
Request received from Client 3: CPU Statistics
Request Approved for Client 3.
Client 3 disconnected.

Request received from Client 0: CPU Statistics
Request Approved for Client 0.
Client 0 disconnected.

Request received from Client 1: CPU Statistics
Request Approved for Client 1.
Client 1 disconnected.

Request received from Client 2: CPU Statistics
Request Approved for Client 2.
Client 2 disconnected.

^C
Server shutdown.
Client 0: (127.0.0.1, 49160), Server: (127.0.0.1, 8005)
Client 1: (127.0.0.1, 49176), Server: (127.0.0.1, 8005)
Client 2: (127.0.0.1, 49186), Server: (127.0.0.1, 8005)
Client 3: (127.0.0.1, 49192), Server: (127.0.0.1, 8005)

 Performance counter stats for 'taskset -c 0 ./server 1':

            470.73 msec task-clock                       #    0.019 CPUs utilized
               250      context-switches                 #  531.087 /sec
                 1      cpu-migrations                   #    2.124 /sec
               456      page-faults                      #  968.702 /sec
     1,432,884,927      cycles                           #    3.044 GHz
       870,788,847      stalled-cycles-frontend          #   60.77% frontend cycles idle
     1,348,922,916      instructions                     #    0.94  insn per cycle
                                                  #    0.65  stalled cycles per insn
       231,830,711      branches                         #  492.489 M/sec
         1,885,560      branch-misses                    #    0.81% of all branches

      24.849493834 seconds time elapsed

       0.117073000 seconds user
       0.354778000 seconds sys
```

Metric	Value
Task Clock	470.73 msec
Context Switches	250
Page Faults	456
CPU Migrations	1
Cycles	3.044 GHz

Client Output & Performance:

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 1 ./client 4
[Client: 0]: Connection Request sent to Server.
[Client: 1]: Connection Request sent to Server.
[Client: 2]: Connection Request sent to Server.
[Client: 3]: Connection Request sent to Server.

[Client: 0]: Request for CPU Statistics sent.
[Client: 1]: Request for CPU Statistics sent.
[Client: 2]: Request for CPU Statistics sent.
[Client: 3]: Request for CPU Statistics sent.
[Client: 3]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560333 clock ticks.
2. PID: 1634, Process Name: (gnome-shell), CPU Time: 161042 clock ticks.

[Client: 0]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560333 clock ticks.
2. PID: 1634, Process Name: (gnome-shell), CPU Time: 161042 clock ticks.

[Client: 1]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560333 clock ticks.
2. PID: 1634, Process Name: (gnome-shell), CPU Time: 161042 clock ticks.

[Client: 2]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560333 clock ticks.
2. PID: 1634, Process Name: (gnome-shell), CPU Time: 161042 clock ticks.


 Performance counter stats for 'taskset -c 1 ./client 4':

              3.51 msec task-clock                       #    0.007 CPUs utilized
                 9      context-switches                 #    2.563 K/sec
                 0      cpu-migrations                   #    0.000 /sec
               135      page-faults                      #   38.452 K/sec
        10,165,214      cycles                           #    2.895 GHz
         8,398,572      stalled-cycles-frontend          #   82.62% frontend cycles idle
         3,762,771      instructions                     #    0.37  insn per cycle
                                                  #    2.23  stalled cycles per insn
           691,475      branches                         #  196.953 M/sec
            36,481      branch-misses                    #    5.28% of all branches

       0.503307731 seconds time elapsed

       0.000742000 seconds user
       0.003712000 seconds sys
```

Metric	Value
Task Clock	3.51 msec
Context Switches	9
Page Faults	135
CPU Migrations	0
Cycles	2.895 GHz

b) **Multithreaded TCP Client-Server**

Server Output & Performance:

We allowed the server to handle 2 clients connections at a single time and pinned the process to one CPU.

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 0 ./server 2
Client 0 connected!
Client 3 connected!
Client 1 connected!
Client 2 connected!
Request received from Client 0: CPU Statistics
Request Approved for Client 0.
Client 0 disconnected.

Request received from Client 2: CPU Statistics
Request Approved for Client 2.
Client 2 disconnected.

Request received from Client 3: CPU Statistics
Request Approved for Client 3.
Client 3 disconnected.

Request received from Client 1: CPU Statistics
Request Approved for Client 1.
Client 1 disconnected.

^C
Server shutdown.
Client 0: (127.0.0.1, 38208), Server: (127.0.0.1, 8005)
Client 1: (127.0.0.1, 38214), Server: (127.0.0.1, 8005)
Client 2: (127.0.0.1, 38228), Server: (127.0.0.1, 8005)
Client 3: (127.0.0.1, 38232), Server: (127.0.0.1, 8005)

 Performance counter stats for 'taskset -c 0 ./server 2':

            438.89 msec task-clock                       #    0.097 CPUs utilized
               256      context-switches                 #  583.291 /sec
                 0      cpu-migrations                   #    0.000 /sec
               454      page-faults                      #    1.034 K/sec
     1,388,942,380      cycles                           #    3.165 GHz
       827,105,649      stalled-cycles-frontend          #   59.55% frontend cycles idle
     1,349,821,218      instructions                     #    0.97  insn per cycle
                                                  #    0.61  stalled cycles per insn
       231,886,017      branches                         #  528.348 M/sec
         2,228,259      branch-misses                    #    0.96% of all branches

       4.535457520 seconds time elapsed

       0.124017000 seconds user
       0.316291000 seconds sys
```

Metric	Value
Task Clock	438.89 msec
Context Switches	256
Page Faults	454
CPU Migrations	0
Cycles	3.165 GHz

Client Output & Performance:

We passed the number of client as 4 connections [total] to the server and pinned the process to one CPU.

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 1 ./client 4
[Client: 0]: Connection Request sent to Server.
[Client: 1]: Connection Request sent to Server.
[Client: 2]: Connection Request sent to Server.
[Client: 3]: Connection Request sent to Server.

[Client: 0]: Request for CPU Statistics sent.
[Client: 1]: Request for CPU Statistics sent.
[Client: 2]: Request for CPU Statistics sent.
[Client: 3]: Request for CPU Statistics sent.
[Client: 0]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560347 clock ticks.
2. PID: 4371, Process Name: (threaded-ml), CPU Time: 161751 clock ticks.

[Client: 2]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560347 clock ticks.
2. PID: 4888, Process Name: (threaded-ml), CPU Time: 161751 clock ticks.

[Client: 3]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560347 clock ticks.
2. PID: 4371, Process Name: (threaded-ml), CPU Time: 161751 clock ticks.

[Client: 1]: Message received from Server: Request Approved :)
Top 2 CPU-consuming processes:
1. PID: 1460, Process Name: (systemd), CPU Time: 560347 clock ticks.
2. PID: 4371, Process Name: (threaded-ml), CPU Time: 161751 clock ticks.


 Performance counter stats for 'taskset -c 1 ./client 4':

              6.97 msec task-clock                       #    0.015 CPUs utilized
                 6      context-switches                 #  861.112 /sec
                 0      cpu-migrations                   #    0.000 /sec
               137      page-faults                      #   19.662 K/sec
        20,384,419      cycles                           #    2.926 GHz
        18,641,032      stalled-cycles-frontend          #   91.45% frontend cycles idle
         3,727,274      instructions                     #    0.18  insn per cycle
                                                  #    5.00  stalled cycles per insn
           683,541      branches                         #   98.101 M/sec
            36,481      branch-misses                    #    5.34% of all branches

       0.463570381 seconds time elapsed

       0.000833000 seconds user
       0.008333000 seconds sys
```

Metric	Value
Task Clock	6.97 msec
Context Switches	6.
Page Faults	137
CPU Migrations	0
Cycles	2.926 GHz

c) **TCP Client-Server using ‘select’**

This is a method of event driven programming. The select call monitors activity on a set of sockets looking for sockets ready for reading, writing, or with an exception condition pending. We created 5 clients, connected to the server, requested for CPU Statistics and did the analysis.

Server Output & Performance:

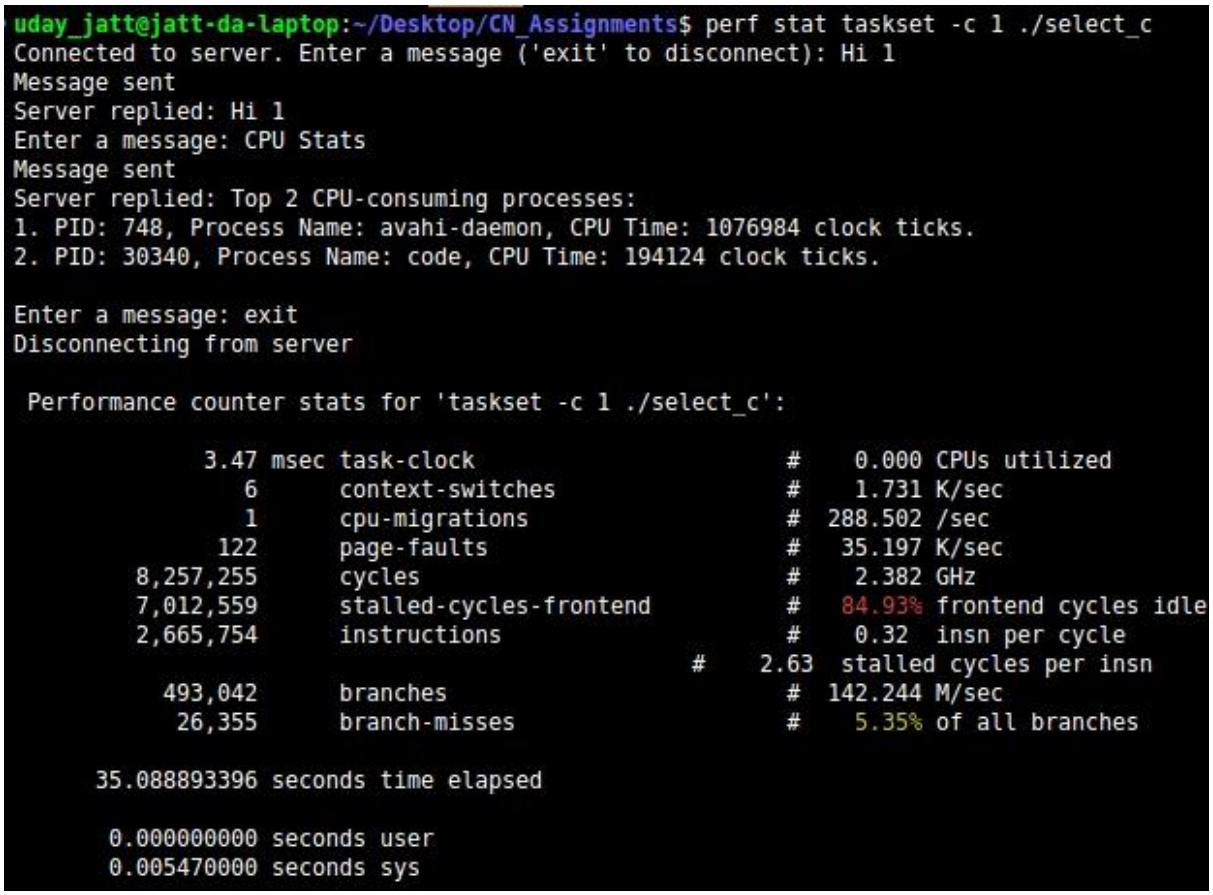
```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 0 ./select_s
Listening on port 8080
New connection, socket fd is 4
Adding to list of sockets as 0
New connection, socket fd is 5
Adding to list of sockets as 1
New connection, socket fd is 6
Adding to list of sockets as 2
New connection, socket fd is 7
Adding to list of sockets as 3
Received: Hi 1
Received: Hi 2
Received: Hi 3
Received: Hi 4
Received: CPU Stats
Received: CPU Stats
Received: CPU Stats
Received: CPU Stats
Host disconnected, IP 127.0.0.1, port 47970
Host disconnected, IP 127.0.0.1, port 47976
Host disconnected, IP 127.0.0.1, port 47986
Host disconnected, IP 127.0.0.1, port 47988
^C
Server shutdown.

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

            468.31 msec task-clock                       #    0.010 CPUs utilized
               131      context-switches                 #  279.728 /sec
                 1      cpu-migrations                   #    2.135 /sec
               197      page-faults                      #  420.660 /sec
     1,405,206,246      cycles                           #    3.001 GHz
       861,875,743      stalled-cycles-frontend          #   61.33% frontend cycles idle
     1,296,070,059      instructions                     #    0.92  insn per cycle
                                                  #    0.66  stalled cycles per insn
       218,544,903      branches                         #  466.666 M/sec
         1,623,687      branch-misses                    #    0.74% of all branches

      46.321687577 seconds time elapsed

       0.100610000 seconds user
       0.368593000 seconds sys
```

Metric	Value
Task Clock	468.31 msec
Context Switches	131
Page Faults	197
CPU Migrations	1
Cycles	3.001 GHz

Client Output & Performance:

We made 4 different clients to connect to the server.

Client 1-



Here is the table for Client 1 values:

Metric	Value
Task Clock	3.47 msec
Context Switches	6
Page Faults	122
CPU Migrations	1
Cycles	2.382 GHz

Client 2-

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 1 ./select_c
Connected to server. Enter a message ('exit' to disconnect): Hi 2
Message sent
Server replied: Hi 2
Enter a message: CPU Stats
Message sent
Server replied: Top 2 CPU-consuming processes:
1. PID: 748, Process Name: avahi-daemon, CPU Time: 1076984 clock ticks.
2. PID: 30340, Process Name: code, CPU Time: 194244 clock ticks.

Enter a message: exit
Disconnecting from server

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

              4.78 msec task-clock                #    0.000 CPUs utilized
                 6      context-switches          #    1.254 K/sec
                 1      cpu-migrations            #  209.013 /sec
               124      page-faults               #   25.918 K/sec
        12,717,882      cycles                    #    2.658 GHz
        11,420,401      stalled-cycles-frontend   #   89.80% frontend cycles idle
         2,744,507      instructions              #    0.22  insn per cycle
                                                  #    4.16  stalled cycles per insn
           506,496      branches                  #  105.865 M/sec
            28,992      branch-misses             #    5.72% of all branches

      36.184331029 seconds time elapsed

       0.000913000 seconds user
       0.005175000 seconds sys
```

Here is the table for Client 2 values:

Metric	Value
Task Clock	4.78 msec
Context Switches	6
Page Faults	124
CPU Migrations	1
Cycles	2.658 GHz

Client 3-

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 1 ./select_c
Connected to server. Enter a message ('exit' to disconnect): Hi 3
Message sent
Server replied: Hi 3
Enter a message: CPU Stats
Message sent
Server replied: Top 2 CPU-consuming processes:
1. PID: 748, Process Name: avahi-daemon, CPU Time: 1076984 clock ticks.
2. PID: 30340, Process Name: code, CPU Time: 194356 clock ticks.

Enter a message: exit
Disconnecting from server

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

              4.76 msec task-clock                #    0.000 CPUs utilized
                 6      context-switches          #    1.261 K/sec
                 1      cpu-migrations            #  210.143 /sec
               123      page-faults               #   25.848 K/sec
        12,697,450      cycles                    #    2.668 GHz
        11,423,581      stalled-cycles-frontend   #   89.97% frontend cycles idle
         2,717,611      instructions              #    0.21  insn per cycle
                                                  #    4.20  stalled cycles per insn
           501,144      branches                  #  105.312 M/sec
            28,524      branch-misses             #    5.69% of all branches

      35.332097634 seconds time elapsed

       0.000941000 seconds user
       0.005297000 seconds sys
```

Here is the table for Client 3 values:

Metric	Value
Task Clock	4.76 msec
Context Switches	6
Page Faults	123
CPU Migrations	1
Cycles	2.668 GHz

Client 4-

```
uday_jatt@jatt-da-laptop:~/Desktop/CN_Assignments$ perf stat taskset -c 1 ./select_c
Connected to server. Enter a message ('exit' to disconnect): Hi 4
Message sent
Server replied: Hi 4
Enter a message: CPU Stats
Message sent
Server replied: Top 2 CPU-consuming processes:
1. PID: 748, Process Name: avahi-daemon, CPU Time: 1076984 clock ticks.
2. PID: 32225, Process Name: DedicatedWorker, CPU Time: 194511 clock ticks.

Enter a message: exit
Disconnecting from server

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

              4.33 msec task-clock                #    0.000 CPUs utilized
                 6      context-switches          #    1.387 K/sec
                 1      cpu-migrations            #  231.098 /sec
               124      page-faults               #   28.656 K/sec
        10,097,734      cycles                    #    2.334 GHz
         8,834,254      stalled-cycles-frontend   #   87.49% frontend cycles idle
         2,694,125      instructions              #    0.27  insn per cycle
                                                  #    3.28  stalled cycles per insn
           498,076      branches                  #  115.104 M/sec
            27,273      branch-misses             #    5.48% of all branches

      35.369785436 seconds time elapsed

       0.001271000 seconds user
       0.005084000 seconds sys
```

Here is the table for Client 4 values:

Metric	Value
Task Clock	4.33 msec
Context Switches	6
Page Faults	124
CPU Migrations	1
Cycles	2.334 GHz

Client Output Averages:

The averages for each metric across the four clients are:

Metric	Average Value
Task Clock	4.335 msec
Context Switches	6
Page Faults	123.25
CPU Migrations	1
Cycles	2.510 GHz

Summary:

Server:

1. Task Clock:

- **Multithreaded Server:** Achieves the lowest task clock (438.89 msec), indicating faster overall processing. This is expected because the multithreaded approach allows concurrent client handling, reducing total time.
- **Select (Event-Driven):** Has a slightly higher task clock (468.31 msec) than the multithreaded model but still performs better than the single-threaded model. The event-driven method efficiently monitors multiple clients but has some overhead for handling events.
- **Single Threaded Server:** Takes the longest time (470.73 msec) since it processes requests sequentially, which increases the total time to handle multiple clients.

2. Context Switches:

- **Select (Event-Driven):** Performs significantly fewer context switches (131) compared to the multithreaded (256) and single-threaded (250) models. This is because select() handles all connections within a single thread, reducing the need for frequent context switches.
- **Multithreaded Server:** Experiences more context switches (256) due to thread management. Each client is handled by a separate thread, which requires the CPU to switch between threads frequently.
- **Single Threaded Server:** Also has a high number of context switches (250) as it repeatedly switches between the client process and the server process while handling requests sequentially.

3. Page Faults:

- **Select (Event-Driven):** Has the fewest page faults (197), indicating more efficient memory management. The event-driven model avoids excessive memory allocation and swapping, which often happens with thread-heavy applications.
- **Single Threaded and Multithreaded Servers:** Both experience a high number of page faults (456 and 454 respectively). This can be attributed to higher memory usage and context switching, especially in multithreaded models that require more resources for thread management.

4. CPU Migrations:

- **Single Threaded & Select (Event-Driven):** Each experienced 1 CPU migration, likely due to load balancing when the process moved between cores.
- **Multithreaded Server:** No CPU migrations were recorded, likely because the process was pinned to one CPU as per the design, or the system had sufficient resources to avoid migrations.

5. Cycles:

- **Multithreaded Server:** Operates at the highest frequency (3.165 GHz) as the CPU is dynamically boosting its clock speed to handle multiple concurrent threads.
- **Single Threaded Server:** Operates at 3.044 GHz, which is slightly lower due to the absence of thread management, but the server still needs to handle frequent context switches.
- **Select (Event-Driven):** Operates at the lowest frequency (3.001 GHz), reflecting the efficient use of CPU resources in the event-driven model. The absence of thread management and reduced memory contention contribute to lower CPU usage.

Hypothesis (Server-Side):

- The **multithreaded server** performs the fastest in terms of overall task completion, but this comes at the cost of higher context switches, page faults, and CPU usage due to thread management.
- The **select (event-driven) server** strikes a balance between efficiency and speed, with fewer context switches and page faults. It is a better choice for scenarios where CPU and memory efficiency are prioritized.
- The **single-threaded server** is the simplest design but is inefficient when handling multiple clients due to sequential processing, leading to longer task clocks and more context switches.

Clients:

1. Task Clock:

- **Single Threaded Client:** Has the lowest task clock (3.51 msec), meaning each client completes its task faster in this model. This is expected since there is no concurrency, and each client is processed in isolation without contention for resources.
- **Multithreaded Client:** Shows the highest task clock (6.97 msec), which indicates that client requests take longer. The overhead of managing multiple threads and contention between them slows down the processing speed for each individual client.
- **Select (Event-Driven):** Offers a balanced task clock (4.335 msec), faster than the multithreaded model but slower than the single-threaded client. This is due to the efficient event-driven mechanism, which allows the server to monitor multiple clients without the overhead of multithreading.

2. Context Switches:

- **All Clients (Except Single Threaded):** The multithreaded and event-driven clients experience similar context switches (6 on average), as they both efficiently handle connections with minimal overhead.
- **Single Threaded Client:** Has more context switches (9), likely because it alternates between processing each client in a non-concurrent manner, resulting in more frequent switches between client and server processes.

3. Page Faults:

- **Select (Event-Driven):** Again, performs better here with the fewest page faults (123.25). The event-driven nature minimizes memory usage and contention, resulting in fewer page faults.
- **Multithreaded Client:** Has slightly more page faults (137) due to the memory overhead of managing multiple threads.
- **Single Threaded Client:** Falls in between (135), reflecting moderate memory usage without the complexity of thread management.

4. CPU Migrations:

- **Multithreaded & Single Threaded Clients:** Show no CPU migrations, as they were pinned to the CPU or had sufficient resources without requiring migration.

- **Select (Event-Driven):** Shows 1 CPU migration, possibly due to load balancing when the process switched cores during execution.

5. Cycles:

- **Multithreaded Client:** Operates at the highest frequency (2.926 GHz), driven by the increased demand from handling multiple threads concurrently.
- **Single Threaded Client:** Operates at 2.895 GHz, reflecting less CPU demand compared to the multithreaded model.
- **Select (Event-Driven):** Operates at the lowest frequency (2.5105 GHz), indicating the most efficient CPU usage due to the event-driven design's lower resource contention.

Hypothesis (Client-Side):

- The **single-threaded client** benefits from faster task clocks and fewer page faults due to sequential processing, but the lack of concurrency can lead to inefficiencies in handling multiple connections simultaneously.
- The **multithreaded client** suffers from higher task clocks and CPU usage due to the overhead of managing threads. It is suited for parallel tasks but at the cost of efficiency.
- The **select (event-driven) client** offers a good compromise with moderate task clocks, fewer page faults, and efficient CPU usage. It is better for scenarios where multiple connections need to be handled with minimal resource overhead.

Conclusion:

For the server-side, the single-threaded model is the least efficient, exhibiting the longest task clock (470.73 msec) due to its sequential processing of requests, which results in high context switches (250) and page faults (456). The multithreaded server performs better with a task clock of 438.89 msec but incurs significant overhead from managing multiple threads, reflected in its higher context switches (256) and similar page faults (454). In contrast, the select (event-driven) server strikes an optimal balance, offering a competitive task clock (468.31 msec) while exhibiting the fewest context switches (131) and page faults (197), making it more efficient for handling multiple clients.

On the client side, the single-threaded client again performs the fastest with an average task clock of 3.51 msec, but its sequential nature limits its ability to handle concurrent requests effectively. The multithreaded client shows a slower task clock (6.97 msec) due to overhead from thread management, while also having similar page faults (137) and context switches (6). The select (event-driven) client provides a middle ground with an average task clock of 4.335 msec, fewer page faults (123.25), and efficient CPU usage. Overall, the select model demonstrates superior scalability and efficiency across both server and client implementations, highlighting its potential for high-demand applications.

----------The End----------