**COMPUTER NETWORK**
**ASSIGNMENT 2**
**REPORT**

**QUESTION 1**

**Server Code**

1) The server sets up a TCP socket and listens for client connections.

**Socket Creation**:

- The **socket() function** creates a TCP socket (with AF_INET for IPv4 and SOCK_STREAM for TCP).
- If the socket creation fails, an error message is printed using **perror(),** and the program exits.

**Binding**:

- The **bind()** function associates the server socket with a specific IP address and port, allowing it to listen on that address.
- If the binding fails, the program prints an error message and exits.

**Listening**:

- The **listen()** function puts the server in a passive listening mode, allowing it to accept incoming client connections. The second argument specifies the maximum number of pending connections in the queue.
- If the server fails to listen, it prints an error message and exits.

2) The server handles 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

**Accepting Client Connections**:

- The **accept()** function waits for incoming client connections on the server's listening socket. When a client connects, a new socket (new_socket) is created specifically for that client connection.
- The 4-tuple created consists of:
  - Server IP
  - Server port
  - Client IP (retrieved from the client connection)
  - Client port (assigned by the client system)

- If the accept() call fails, an error message is printed, and the server continues to listen for the next client.

**Thread Creation**:

- For each new client, the server creates a new thread using **pthread_create().** This allows the server to handle multiple clients concurrently.
- The new_socket is passed to the thread by allocating memory for the socket pointer and passing it as an argument to the **handle_client() function.**

**Thread Detachment**:

- **pthread_detach()** is called to ensure the thread cleans up its resources when it finishes. This allows the server to handle multiple client threads without manual thread management.

3) The original server socket continues to listen on the same listening port for newer incoming client connections.

- The server is placed in an infinite loop, allowing it to call accept() and handle new client connections continuously.
- Each time accept() is called, the server waits for a new client connection on the same listening port (8080). Once a client connects, the server creates a new socket (new_socket) for that specific connection, but the original socket (server_fd) continues listening for additional clients.

**Non-Blocking Server**:

- After each client connection is accepted and handed off to a new thread via **pthread_create(),** the server immediately returns to the top of the loop and waits for the next client connection.
- This allows the original server socket (server_fd) to remain open on the same port and handle new incoming connections while other clients are being processed in separate threads.

4) 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

    - This functionality is handled by the **get_process_info()** function, which reads the /proc directory to find running processes and gathers information such as process name, PID, and CPU usage in user and kernel modes. It calculates the total CPU time (user time + kernel time) for each process and identifies the top two CPU-consuming processes.

**CLIENT**

1) The client creates a socket and initiates the TCP connection. The client process supports initiating "n" concurrent client connection requests, where "n" is passed as a program argument.
   - The client creates a socket using **socket()** to establish a **TCP connection**. If the creation fails, it prints an error message and exits the thread.
   - The client supports initiating **"n" concurrent connection requests** by creating multiple threads. The number of client threads is determined by the command-line argument passed when executing the program. Each thread runs the **client_thread()** function, which handles the connection and communication with the server.

2) 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.
   - After the client establishes a TCP connection, it sends a message to the server, indicating a request for information about the top two CPU-consuming processes.

3) The client prints this information & closes the connection.
   - The **read()** function is used to receive the server's response, which is then printed to the console.
   - After the response is printed, the client closes the connection using **close()**, indicating that the communication with the server is complete.

NOTE: taskset was used for client and server code to pin task to specific CPUs

## QUESTION 2

NOTE: Readings were taken after using taskset to pin tasks to specific CPUs.

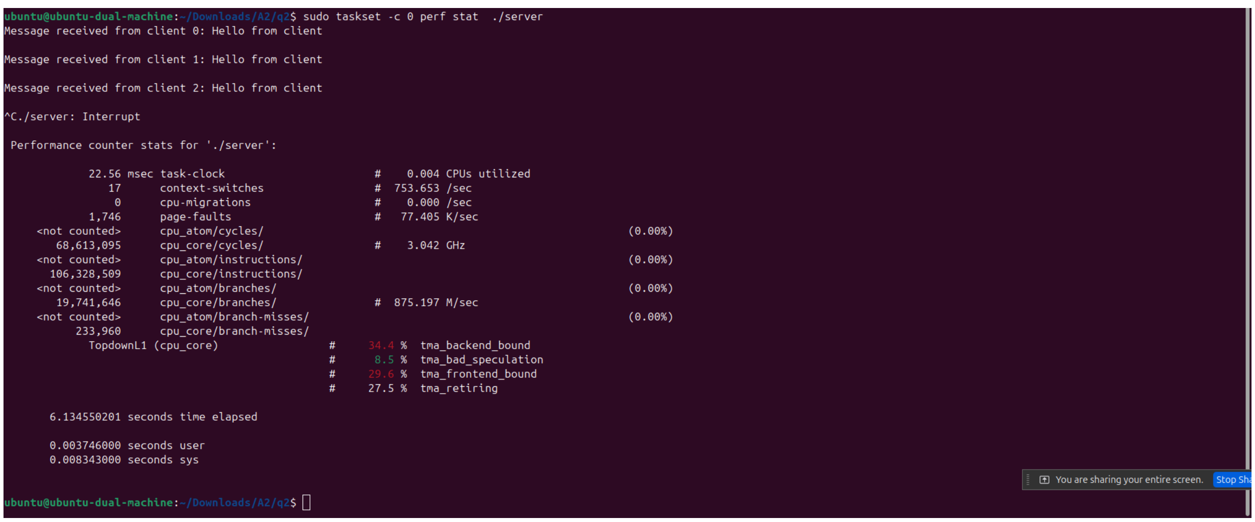
### (a) Single-threaded TCP client-server

```
ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$ sudo taskset -c 0 perf stat ./server
Server is now listening for incoming connections on port 8002
Message received from client: Requesting top CPU processes
Message received from client: Requesting top CPU processes
Message received from client: Requesting top CPU processes
Client disconnected.
^X^C./server: Interrupt

 Performance counter stats for './server':

            15.39 msec task-clock                       #    0.002 CPUs utilized
                7      context-switches                 #  454.911 /sec
                0      cpu-migrations                   #    0.000 /sec
              143      page-faults                      #    9.293 K/sec
     <not counted>      cpu_atom/cycles/                                                      (0.00%)
       26,622,471      cpu_core/cycles/                 #    1.730 GHz
     <not counted>      cpu_atom/instructions/                                                (0.00%)
       52,089,912      cpu_core/instructions/
     <not counted>      cpu_atom/branches/                                                    (0.00%)
        9,614,604      cpu_core/branches/               #  624.826 M/sec
     <not counted>      cpu_atom/branch-misses/                                               (0.00%)
           68,099      cpu_core/branch-misses/
            TopdownL1 (cpu_core)                 #     30.7 %  tma_backend_bound
                                                 #      6.4 %  tma_bad_speculation
                                                 #     29.4 %  tma_frontend_bound
                                                 #     33.5 %  tma_retiring

      6.487866455 seconds time elapsed

      0.003354000 seconds user
      0.012300000 seconds sys


ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$
```

**Performance counter stats for './server'**

| | | |
|---|---|---|
| 15.39 msec | task-clock | # 0.002 CPUs utilized |
| 7 | context-switches | # 454.911 /sec |
| 0 | cpu-migrations | # 0.000 /sec |
| 143 | page-faults | # 9.293 K/sec |
| 26,622,471 | cpu_core/cycles/ | # 1.730 GHz |
| 52,089,912 | cpu_core/instructions/ | |
| 9,614,604 | cpu_core/branches/ | # 624.826 M/sec |
| 68,099 | cpu_core/branch-misses/ | |
| 6.487866455 seconds | time elapsed | |
| 0.003354000 seconds | user | |
| 0.012300000 seconds | sys | |

## (b) Multi-threaded TCP client-server



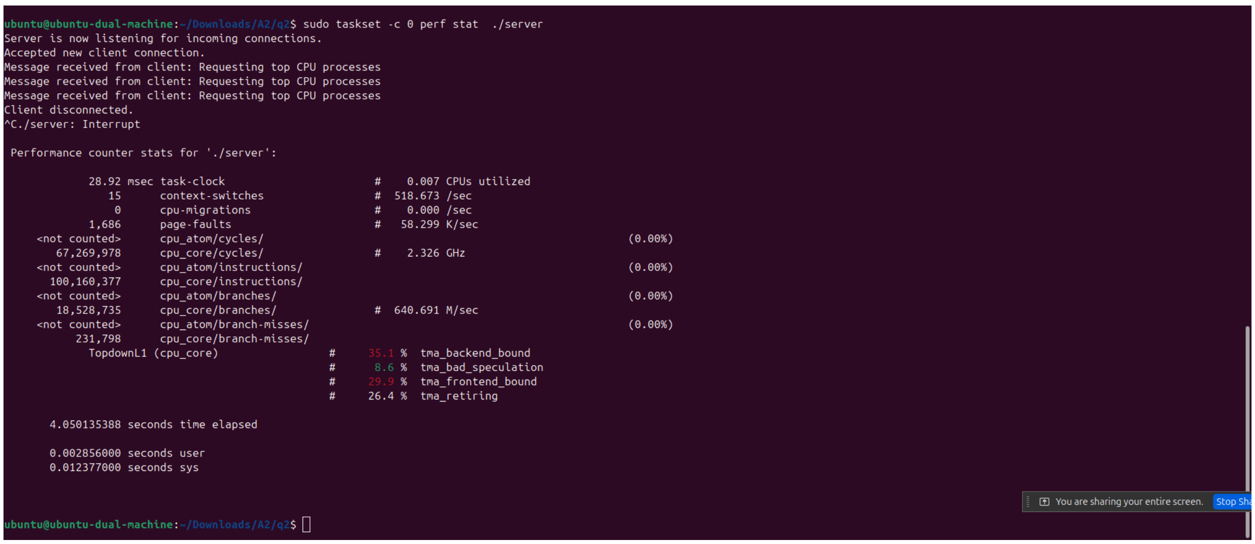
```
ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$ sudo taskset -c 0 perf stat  ./server
Message received from client 0: Hello from client

Message received from client 1: Hello from client

Message received from client 2: Hello from client

^C./server: Interrupt

 Performance counter stats for './server':

             22.56 msec task-clock                       #    0.004 CPUs utilized
                17      context-switches                 #  753.653 /sec
                 0      cpu-migrations                   #    0.000 /sec
             1,746      page-faults                      #   77.405 K/sec
     <not counted>      cpu_atom/cycles/                                                        (0.00%)
        68,613,095      cpu_core/cycles/                 #    3.042 GHz
     <not counted>      cpu_atom/instructions/                                                  (0.00%)
       106,328,509      cpu_core/instructions/
     <not counted>      cpu_atom/branches/                                                      (0.00%)
        19,741,646      cpu_core/branches/               #  875.197 M/sec
     <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
           233,960      cpu_core/branch-misses/
             TopdownL1 (cpu_core)                 #     34.4 %  tma_backend_bound
                                                  #      8.5 %  tma_bad_speculation
                                                  #     29.6 %  tma_frontend_bound
                                                  #     27.5 %  tma_retiring

       6.134550201 seconds time elapsed

       0.003746000 seconds user
       0.008343000 seconds sys

ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$
```

**Performance counter stats for './server'**

| | | |
|---|---|---|
| 22.56 msec | task-clock | # 0.004 CPUs utilized |
| 17 | context-switches | # 753.653 /sec |
| 0 | cpu-migrations | # 0.000 /sec |
| 1,746 | page-faults | # 77.405 K/sec |
| 68,613,095 | cpu_core/cycles/ | # 3.042 GHz |
| 106,328,509 | cpu_core/instructions/ | |
| 19,741,646 | cpu_core/branches/ | # 875.197 M/sec |
| 233,960 | cpu_core/branch-misses/ | |

| | |
|---|---|
| 6.13455021 seconds | time elapsed |
| | |
| 0.003746000 seconds | user |
| 0.008343000 seconds | sys |

### (c) TCP client-server using “select”



```
ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$ sudo taskset -c 0 perf stat  ./server
Server is now listening for incoming connections.
Accepted new client connection.
Message received from client: Requesting top CPU processes
Message received from client: Requesting top CPU processes
Message received from client: Requesting top CPU processes
Client disconnected.
^C./server: Interrupt

 Performance counter stats for './server':

            28.92 msec task-clock                       #    0.007 CPUs utilized
               15      context-switches                 #  518.673 /sec
                0      cpu-migrations                   #    0.000 /sec
            1,686      page-faults                      #   58.299 K/sec
     <not counted>      cpu_atom/cycles/                                                        (0.00%)
       67,269,978      cpu_core/cycles/                 #    2.326 GHz
     <not counted>      cpu_atom/instructions/                                                  (0.00%)
      100,160,377      cpu_core/instructions/
     <not counted>      cpu_atom/branches/                                                      (0.00%)
       18,528,735      cpu_core/branches/               #  640.691 M/sec
     <not counted>      cpu_atom/branch-misses/                                                 (0.00%)
          231,798      cpu_core/branch-misses/
            TopdownL1 (cpu_core)                 #     35.1 %  tma_backend_bound
                                                 #      8.6 %  tma_bad_speculation
                                                 #     29.9 %  tma_frontend_bound
                                                 #     26.4 %  tma_retiring


       4.050135388 seconds time elapsed

       0.002856000 seconds user
       0.012377000 seconds sys


ubuntu@ubuntu-dual-machine:~/Downloads/A2/q2$
```

**Performance counter stats for './server'**

| | |
|---|---|
| 28.92 msec | task-clock # 0.007 CPUs utilized |
| 15 | context-switches # 518.673 /sec |
| 0 | cpu-migrations # 0.000 /sec |
| 1,686 | page-faults # 58.299 K/sec |
| 67,269,978 | cpu_core/cycles/ # 2.326 GHz |
| 100,160,377 | cpu_core/instructions/ |
| 18,528,735 | cpu_core/branches/ # 640.691 M/sec |
| 231,798 | cpu_core/branch-misses/ |
| 4.050153388 seconds | time elapsed |
| 0.002850000 seconds | user |
| 0.012377000 seconds | sys |

1. Task Clock and CPUs Utilized:

- **Single-Threaded Server**:
  - Task clock: 15.39 msec

- CPUs utilized: 0.002
- **Multi-threaded Server**:
  - Task clock: 22.56 msec
  - CPUs utilized: 0.004
- **Server using Select**:
  - Task clock: 28.92 msec
  - CPUs utilized: 0.007

The **Server using Select** spent the longest amount of time actively executing instructions on the CPU, while the **Single-Threaded Server** spent the least time. The **Multi-threaded Server** shows moderate execution time, which aligns with its more complex structure compared to the single-threaded version.

**2.** Context Switches (switches between kernel and user space):

- **Single Thread**: 7 context switches
- **MultiThread**: 17 context switches
- **Select-based**: 15 context switches
- The **single-threaded** server has the fewest context switches, as expected. It handles requests sequentially in a single process, leading to minimal task switching.
- The **multi-threaded** server has the most context switches because each thread may need to switch contexts frequently to handle multiple requests concurrently.
- The **select-based** server has context switches slightly lower than the multi-threaded one. The select-based model, though handling multiple connections, typically does so within a single thread, reducing the need for frequent switching but still causing more than a single-threaded model.

3. Page Faults:

- **Single-Threaded Server**: 143 (9.293 K/sec)
- **Multi-threaded Server**: 1,746 (77.405 K/sec)
- **Server using Select**: 1,686 (58.299 K/sec)

- The **single-threaded** server has the fewest page faults, probably because it doesn't handle many requests simultaneously and thus has fewer memory access conflicts.
- Both **multi-threaded** and **select-based** servers have significantly higher page faults. This makes sense, as both handle multiple requests simultaneously, potentially needing more memory allocation, leading to more page faults.

4. CPU Core Cycles:

- **Single-Threaded Server**: 26,622,471 cycles (1.730 GHz)
- **Multi-threaded Server**: 68,613,095 cycles (3.042 GHz)
- **Server using Select**: 67,269,978 cycles (2.326 GHz)

The **multi-threaded** server uses the most CPU cycles, closely followed by the select-based server. This can be explained by both servers having more tasks to manage in parallel (multiple threads or connections). The **single-threaded** server, with fewer tasks at a time, uses fewer cycles overall.

5. Instructions:

- **Single-Threaded Server**: 52,089,912 instructions
- **Multi-threaded Server**: 106,328,509 instructions
- **Server using Select**: 100,160,377 instructions

The **Multi-threaded Server** executes more than double the number of instructions compared to the **Single-Threaded Server**, as expected due to the complexity of handling multiple threads. The **Server using Select** executes a similar number of instructions to the multi-threaded server.

6. Branch Misses:

- **Single-Threaded Server**: 68,099 branch misses
- **Multi-threaded Server**: 233,960 branch misses
- **Server using Select**: 231,798 branch misses

**Branch misses** represent inefficiency in CPU operations. The multi-threaded and select-based servers both have significantly more branch misses, which is expected because they handle more complex and parallelized logic, increasing the chance of mispredictions. The single-threaded server, with simpler flow control, has far fewer branch misses

7. User and Sys Time:

- **Single-Threaded Server**:
  - User: 0.003354 seconds
  - Sys: 0.012300 seconds
- **Multi-threaded Server**:
  - User: 0.003746 seconds
  - Sys: 0.008343 seconds
- **Server using Select**:
  - User: 0.002850 seconds
  - Sys: 0.012377 seconds

8. CPU Migrations: Zero for all the three as taskset pinned the task to specific CPUs

The **Single-Threaded Server** and **Select Server** spend a slightly higher amount of time in system calls (sys time), while the **Multi-threaded Server** has more time in user space (likely due to managing multiple threads).

Summary:

- ➔ **Single Threaded**: Best for simple, low-concurrency applications where system resources (e.g., memory and CPU cycles) must be minimized.
- ➔ **Multi-Threaded**: Suitable for high-load applications requiring parallel processing, though it incurs a higher system overhead.
- ➔ **Select-based**: Best for moderately loaded servers where efficient handling of multiple connections is needed without the heavy overhead of multi-threading.