

# CN Assignment - 2

Sidhartha Garg(2022499), Snehil Jaiswal(2022503)

---

## Server Code Overview

The server is designed to:

- Listen on port 8080 for incoming TCP connections(can use any other port but will have to deactivate the firewall)
- Handle multiple clients concurrently using threads.
- Send information about the top two CPU-consuming processes to each client.

### Key Components:

#### 1. Process Info Collection:

The server fetches information about the top two CPU-consuming processes by scanning the `/proc` filesystem, extracting the process ID (PID), name, user time, and kernel time.

#### 2. Multithreading:

When a client connects, a new thread is created for each client using the `pthread_create` function. The main thread continues to listen for new connections while the worker threads handle the individual clients. Each client thread collects the required process information and sends it to the client.

#### 3. Concurrency:

The server uses the `pthread` library to ensure that multiple clients can be served concurrently. Each client is assigned a separate thread to handle the process without blocking other incoming connections.

### Compiling the Server:

```
gcc server.c -o server -lpthread
```

## Client Code Overview

The client code:

- Connects to the server, requests information about the top CPU-consuming processes, and prints the received information.
- The client is capable of connecting to the server multiple times based on the command-line argument provided.

### Usage:

```
./client <IP address> <Port> <Number of clients>
```

## Execution and Results

- **Server Execution:**

The server is launched, and it listens for incoming connections on port 8080. Each new client triggers a new thread for handling the connection.

The server output displays the top two CPU-consuming processes for each client request, showing both `systemd` and `python3` processes with their respective user and kernel times.

```
sidhartha@sidhartha-VirtualBox:~/Desktop$ taskset -c 1 ./client 172.20.45.228 8080 2
Client 1 received:
Top CPU-consuming process 1:
PID: 1
Name: (systemd)
User time: 39933
Kernel time: 3938

Top CPU-consuming process 2:
PID: 598
Name: (python3)
User time: 16853
Kernel time: 11214

Client 2 received:
Top CPU-consuming process 1:
PID: 1
Name: (systemd)
User time: 39933
Kernel time: 3938

Top CPU-consuming process 2:
PID: 598
Name: (python3)
User time: 16853
Kernel time: 11214
```

- **Client Execution:**

In the client output, we can see that each client receives detailed information about the top two CPU-consuming processes from the server. The client was executed with multiple instances using a command like:

```
taskset -c 1 ./client <IP> <Port> <Number of clients>
```

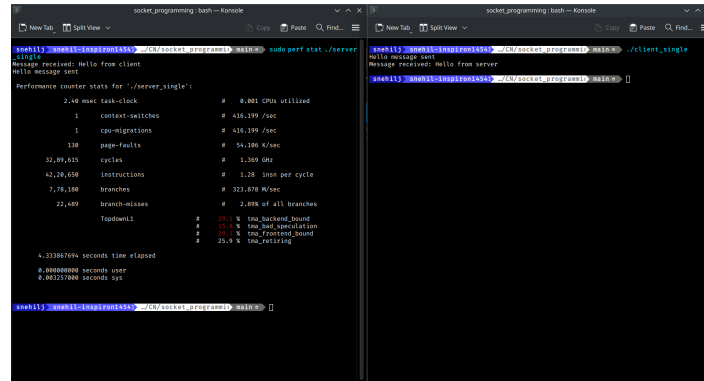
The results show that the server successfully handles multiple clients concurrently, providing consistent CPU usage data to each client.

## 2

### 2.1 (a)

The performance stats are shown in Figure 1:

- **Task Clock: 2.40msec:** The server task took approximately 2.40 milliseconds of wall-clock time to complete. This suggests that the server code was very lightweight.
- **Context Switches: 1:** The server had only 1 context switch during its execution. This indicates minimal switching between user-space and kernel-space. **Context Switches were varying on various occasions. However the number stayed low ranging from 1-3.**
- **CPU Migrations: 1:** The process was migrated once between CPU cores, which is low and indicates little overhead related to CPU resource allocation.
- **Page faults (130):** Although generally memory use is probably well-optimized, this is quite low, indicating the program has to fetch some data from disk. **Page Faults also varied through various executions.**
- **Top-down Metrics (TMA):**
  - **TMA Backend Bound (29.1%):** This indicates 29.1% of time was spent waiting on the backend (e.g., memory accesses)
  - **TMA Bad Speculation (15.8%):** This suggests that a moderate amount of time was lost due to incorrect speculative execution.
  - **TMA Frontend Bound (29.3%):** Time spent waiting for instruction fetches.
  - **TMA Retiring (25.9%):** This shows how much time was spent actually retiring instructions (doing useful work)



```
socket_programming: bash — Konsole
New Tab Split View Copy Paste Find...
ssh@111: ~$ cd /socket_programming
ssh@111: ~$ sudo perf stat ./server
Message received: Hello from client
Hello message sent
Performance counter stats for './server_single':
2.40 msec task-clock # 0.001 CPUs utilized
1 context-switches # 435.189 /sec
1 cpu-migrations # 435.199 /sec
130 page-faults # 54.186 K/sec
32,497,615 cycles # 1.369 GHz
42,28,658 instructions # 1.28 insn per cycle
7,18,188 branches # 323.878 M/sec
22,489 branch-misses # 2.80% of all branches
Topdown:1 #
29.1 % time_backend_bound
15.8 % time_bad_speculation
29.3 % time_frontend_bound
25.9 % time_retiring
4.333867694 seconds time elapsed
0.000000000 seconds user
0.002257900 seconds sys
ssh@111: ~$

socket_programming: bash — Konsole
New Tab Split View Copy Paste Find...
ssh@111: ~$ cd /socket_programming
ssh@111: ~$ ./client_single
Hello message sent
Message received: Hello from server
ssh@111: ~$
```

Figure 1: Performance stats for Single-Threaded TCP connection

## 2.2 (b)

We used the code for part 1 to setup a concurrent multi-threaded connection. The performance stats are shown in Figure 2:

- **Task Clock: 39.28 msec:** The task clock has increased significantly from previous runs. This is due to the multi-threaded server handling multiple client requests (5 clients), which requires more CPU time and processing.
- **Context Switches (22):** The number of context switches (22) is higher compared to earlier results (5 context switches for fewer threads). This increase is expected with more threads handling multiple clients concurrently, as the kernel must switch between them.
- **CPU Migrations (5):** There are now 5 CPU migrations, which indicate the threads were likely moved between different CPU cores for load balancing
- **Page Faults (143):** The page faults are slightly higher than in the previous examples, which is expected due to more memory usage and thread handling.
- **Top-down Metrics (TMA):**
  - TMA Backend Bound : 22.3%
  - TMA Bad Speculation : 6.3%
  - TMA Frontend Bound : 35.4%
  - TMA Retiring : 36.1%
- **Improvements shown:**
  - **Increased Efficiency:** The IPC (1.78) and the TMA Retiring (36.1%) show that the server handled the additional workload quite efficiently
  - **Low Branch Miss Rate:** Despite the large number of branches, the branch miss rate remains low, contributing to the overall efficiency.

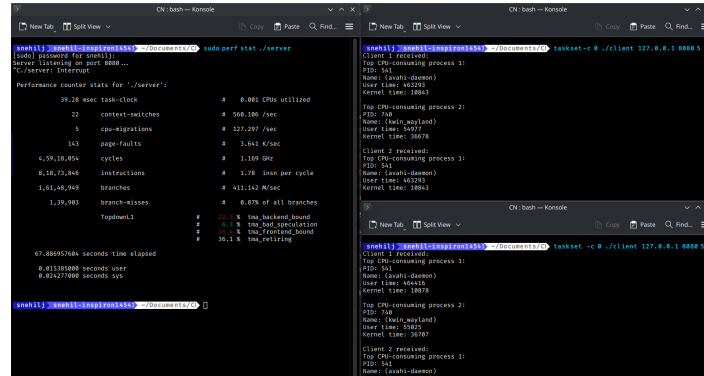


Figure 2: Performance stats for Multi-Threaded TCP connection

## 2.3 (c)

- **Task Clock (20.64 msec):** The task clock is relatively low, reflecting the efficient handling of multiple clients. This suggests the server is responsive and the select-based model works efficiently with a small load.
- **Context Switches (10) and CPU Migrations (1):** There are a few context switches (10) and only 1 CPU migration. This is efficient, as the server isn't being frequently switched between cores, and the system isn't under significant stress.
- **Page Faults (132):** The page fault count is low.
- **Top-down Metrics (TMA):**
  - TMA Backend Bound : 21.1%
  - TMA Bad Speculation : 8.6%
  - TMA Frontend Bound : 34.8%
  - TMA Retiring : 35.5%
- The select() based server design is efficient for handling multiple clients in a non-blocking fashion. The low CPU utilization and high IPC value indicate that the system handles client requests quickly and efficiently.

```
[snehil@snihil-inspiron145410 ~/Documents/Ch] sudo perf stat taskset -c 0 ./server_select
[sudo] password for snehil:
Server listening on port 8080...
New connection from 127.0.0.1:57696
Added to client list at index 0
Received request from client 0: GET_CPU_INFO
Client disconnected: 127.0.0.1:57696
New connection from 127.0.0.1:57708
Added to client list at index 0
Received request from client 0: GET_CPU_INFO
New connection from 127.0.0.1:57722
Added to client list at index 1
Client disconnected: 127.0.0.1:57708
Received request from client 1: GET_CPU_INFO
Client disconnected: 127.0.0.1:57722
New connection from 127.0.0.1:57734
Added to client list at index 0
Received request from client 0: GET_CPU_INFO
New connection from 127.0.0.1:57748
Added to client list at index 1
Client disconnected: 127.0.0.1:57734
Received request from client 1: GET_CPU_INFO
Client disconnected: 127.0.0.1:57748
^Ctaskset: Interrupt

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

    20.64 msec task-clock                #    0.003 CPUs utilized
         10 context-switches            #   484.552 /sec
          1 cpu-migrations               #   48.455 /sec
        132 page-faults                 #    6.396 K/sec
  2,37,05,199 cycles                     #    1.149 GHz
  4,28,10,461 instructions               #    1.81 insn per cycle
    84,47,857 branches                   #   409.343 M/sec
     65,209 branch-misses                #    0.77% of all branches
        TopdownL1                        #   21.1 % tma_backend_bound
                                         #    8.6 % tma_bad_speculation
                                         #   34.8 % tma_frontend_bound
                                         #   35.5 % tma_retiring

 6.104673311 seconds time elapsed

 0.008989000 seconds user
 0.012430000 seconds sys
```

Figure 3: Concurrent server using select()