CODE ANALYSIS

Server Code

1. Include Headers

- Includes standard libraries for functionality:
    - I/O operations (`stdio.h`), memory allocation (`stdlib.h`), string manipulation (`string.h`), network programming (`unistd.h`, `arpa/inet.h`), directory handling (`dirent.h`), and multithreading (`pthread.h`).

2. Define the `Process` Struct

- Defines a `Process` structure that stores details about each process, including its `name`, `pid` (process ID), `user_time` (CPU time in user mode), `kernel_time` (CPU time in kernel mode), and `total_time` (total CPU time).

3. Function: `read_proc_file`

- Read the `/proc/[pid]/stat` file to extract process information.
- Opens the file and extracts process ID, process name, user time, and kernel time.
- Removes parentheses from the process name and calculates the `total_time` as the sum of `user_time` and `kernel_time`.
- Returns `0` on success, `-1` if the file cannot be opened.

4. Function: `get_top_processes`

- Opens the `/proc` directory and iterates over all entries to identify processes.
- Uses `read_proc_file` to read information for each process and stores it in an array.
- Performs a bubble sort to arrange processes in descending order based on `total_time`.
- Constructs a string containing details of the top 2 CPU-consuming processes and stores it in the `buffer`.

5. Function: `handle_client` (Thread Function)

- Accepts a client connection and reads the request sent by the client.
- Calls `get_top_processes` to fetch the top 2 CPU-consuming processes.
- Sends the result back to the client and closes the connection.

6. `main` Function

- Sets up a server using the socket API:
    - Creates a socket, binds it to `INADDR_ANY` (localhost) and port `8005`, and listens for incoming connections.
- Enters an infinite loop to accept client connections:
    - For each connection, creates a new thread using `pthread_create` to handle the client, allowing concurrent handling of multiple clients.
- Uses `pthread_detach` to automatically release resources when the thread exits.

7. Key Functionalities

- Uses multithreading to handle multiple client connections concurrently.
- Uses `bubble sort` to identify the top CPU-consuming processes.
- Implements communication over a socket using the TCP protocol with desired IP (currently coded for loopback) and port `8005`.

8. Error Handling

- Checks for errors during socket creation, binding, listening, and accepting connections, with appropriate error messages and program termination if failures occur.

9. Code Execution Flow

- Start the server program: `main` initializes the server, listens for connections, and creates threads for each client.
- Each thread (`handle_client`) processes the client request, retrieves the top 2 CPU-consuming processes, and responds to the client.

This explanation is suitable for documentation and provides a comprehensive overview of the code's functionality and workflow.

Client Code

1. **Purpose**: The client establishes a connection to the server, sends a request for CPU information, and displays the server's response. Multiple clients are handled using multithreading.
2. **Command-Line Argument**:
    - The program takes one argument: the number of clients to create. Example usage: `./client 3`.
3. **Socket Creation**:
    - `socket(AF_INET, SOCK_STREAM, 0)`: Creates a TCP socket.
4. **Server Address Setup**:
    - `serv_addr.sin_family = AF_INET`: Specifies IPv4.
    - `serv_addr.sin_port = htons(8005)`: Sets the server port to 8005.

    - `inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)`: Converts the loopback IP address (localhost) to binary form.
5. **Connecting to Server**:
    - `connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))`: Connects the client to the server.
6. **Communication**:
    - `send(sock, client_args->message, strlen(client_args->message), 0)`: Sends a message to the server.
    - `read(sock, buffer, BUFFER_SIZE)`: Reads the server's response into a buffer.
7. **Multithreading**:
    - The program uses `pthread_create` to start a new thread for each client.
    - `pthread_join` ensures the main program waits for all threads to complete.
8. **Memory Management**:
    - `ClientArgs *args = malloc(sizeof(ClientArgs))`: Dynamically allocates memory for each thread's arguments.
    - `free(client_args)`: Frees allocated memory after each thread completes.
9. **Error Handling**:
    - Checks for socket creation, address conversion, and connection failures, printing appropriate error messages.

Command to run the code

- **For single-threaded:**

  Server: perf stat -d /usr/bin/taskset -c 0 ./server

  Client: perf stat -d /usr/bin/taskset -c 1 ./client 1

- **For multi-threaded:**
    - **w/o select:**

      Server: perf stat -d /usr/bin/taskset -c 0 ./server

      Client: perf stat -d /usr/bin/taskset -c 1 ./client 4

    - **with select:**

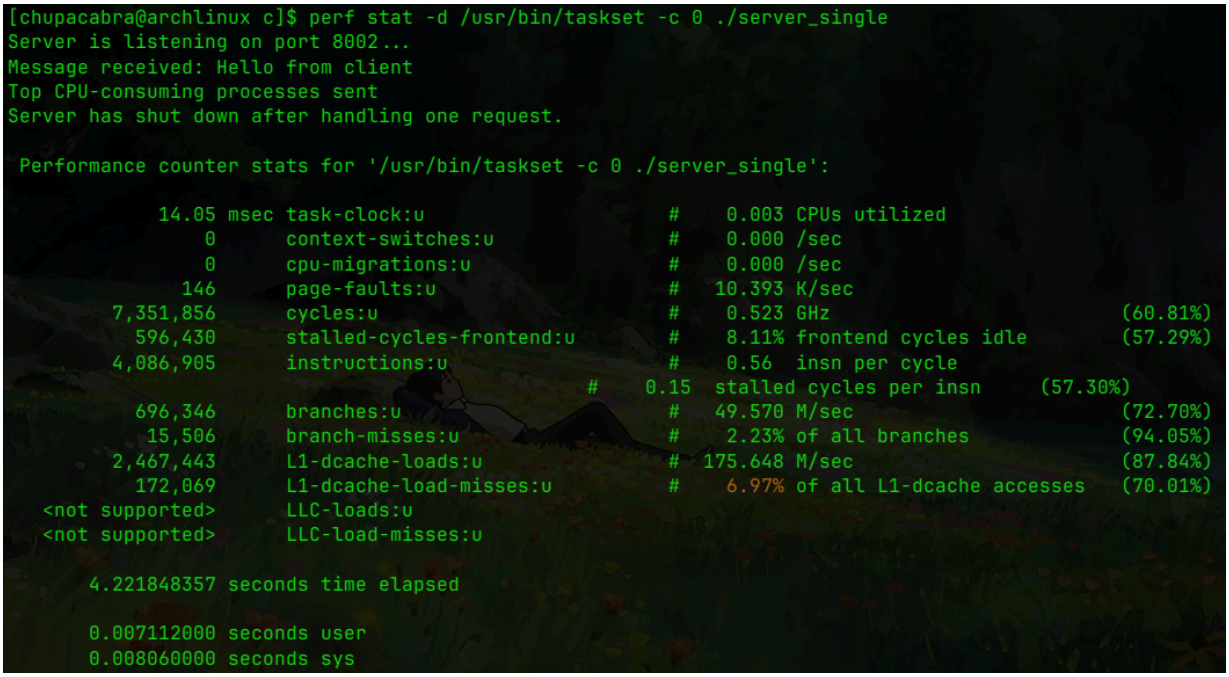
      Server: perf stat -d /usr/bin/taskset -c 0 ./server_select

      Client: perf stat -d /usr/bin/taskset -c 1 ./client 4

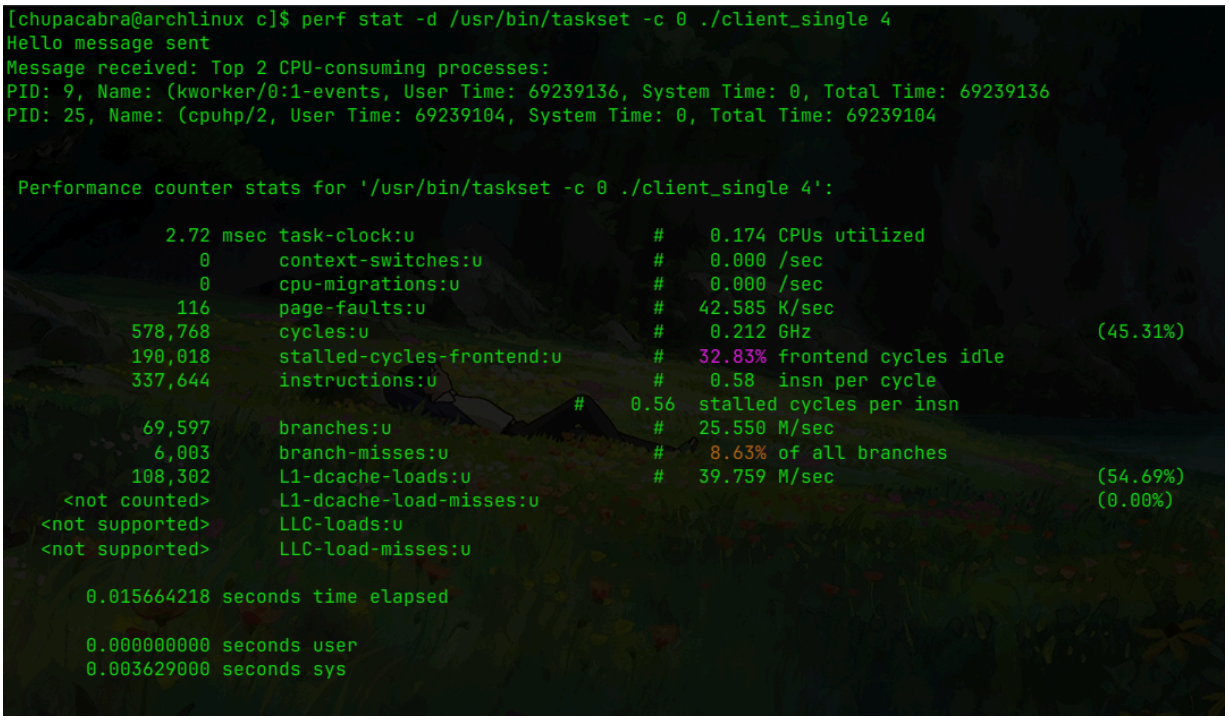
PERFORMANCE ANALYSIS

The observations were made by running Linux natively on the machine without the use of any virtual machine. Running server and client on two virtual machines worked fine but it didn't provide much info in perf.

Single threaded:

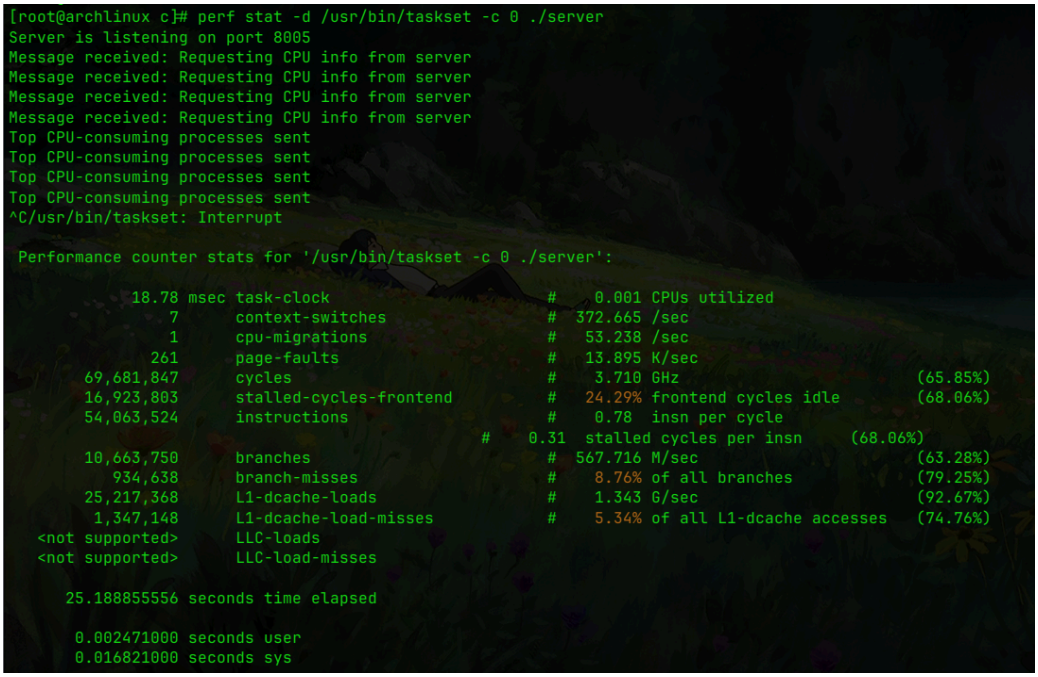


```
[chupacabra@archlinux c]$ perf stat -d /usr/bin/taskset -c 0 ./server_single
Server is listening on port 8002...
Message received: Hello from client
Top CPU-consuming processes sent
Server has shut down after handling one request.

 Performance counter stats for '/usr/bin/taskset -c 0 ./server_single':

            14.05 msec task-clock:u                     #    0.003 CPUs utilized
                0      context-switches:u               #    0.000 /sec
                0      cpu-migrations:u                 #    0.000 /sec
              146      page-faults:u                    #   10.393 K/sec
        7,351,856      cycles:u                         #    0.523 GHz                      (60.81%)
          596,430      stalled-cycles-frontend:u        #    8.11% frontend cycles idle     (57.29%)
        4,086,905      instructions:u                   #    0.56  insn per cycle
                                                  #    0.15  stalled cycles per insn     (57.30%)
          696,346      branches:u                       #   49.570 M/sec                    (72.70%)
           15,506      branch-misses:u                  #    2.23% of all branches          (94.05%)
        2,467,443      L1-dcache-loads:u                #  175.648 M/sec                    (87.84%)
          172,069      L1-dcache-load-misses:u          #    6.97% of all L1-dcache accesses   (70.01%)
   <not supported>      LLC-loads:u
   <not supported>      LLC-load-misses:u

      4.221848357 seconds time elapsed

      0.007112000 seconds user
      0.008060000 seconds sys
```

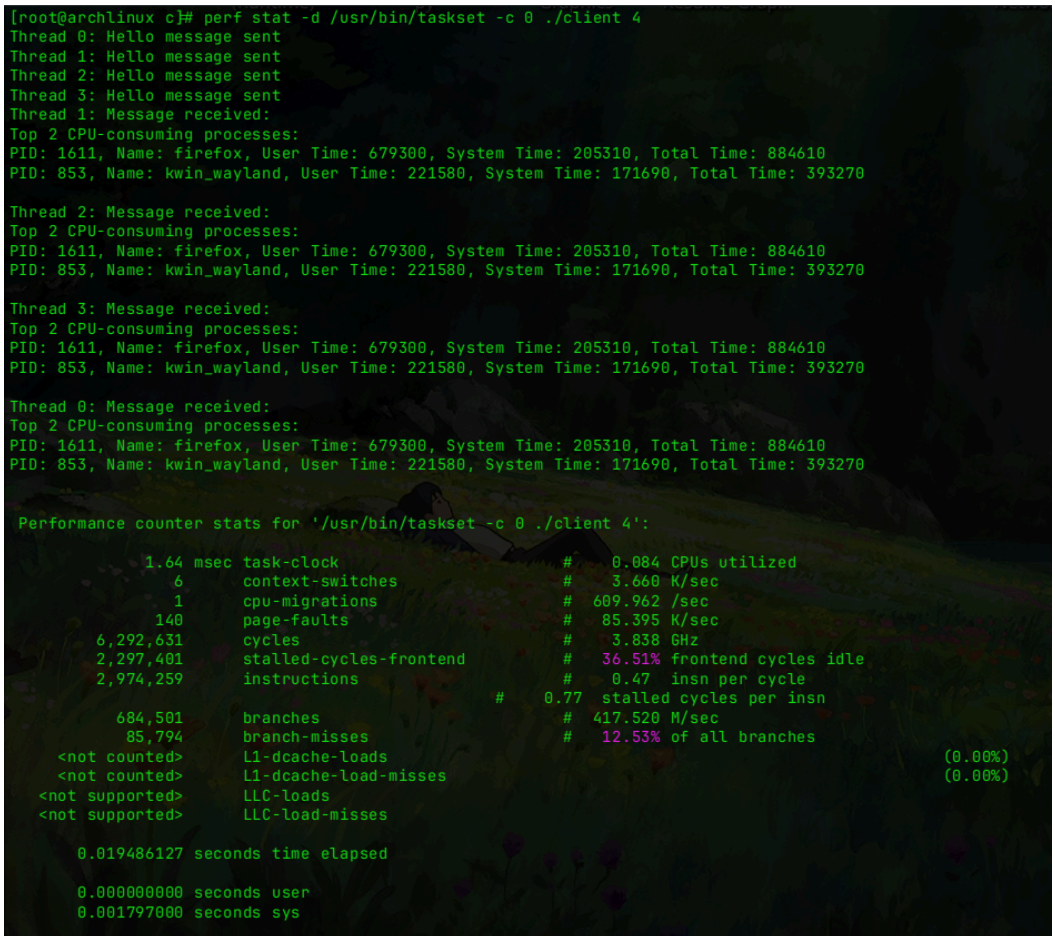


```
[chupacabra@archlinux c]$ perf stat -d /usr/bin/taskset -c 0 ./client_single 4
Hello message sent
Message received: Top 2 CPU-consuming processes:
PID: 9, Name: (kworker/0:1-events, User Time: 69239136, System Time: 0, Total Time: 69239136
PID: 25, Name: (cpuhp/2, User Time: 69239104, System Time: 0, Total Time: 69239104


 Performance counter stats for '/usr/bin/taskset -c 0 ./client_single 4':

             2.72 msec task-clock:u                     #    0.174 CPUs utilized
                0      context-switches:u               #    0.000 /sec
                0      cpu-migrations:u                 #    0.000 /sec
              116      page-faults:u                    #   42.585 K/sec
          578,768      cycles:u                         #    0.212 GHz                      (45.31%)
          190,018      stalled-cycles-frontend:u        #   32.83% frontend cycles idle
          337,644      instructions:u                   #    0.58  insn per cycle
                                                  #    0.56  stalled cycles per insn
           69,597      branches:u                       #   25.550 M/sec
            6,003      branch-misses:u                  #    8.63% of all branches
          108,302      L1-dcache-loads:u                #   39.759 M/sec                    (54.69%)
    <not counted>      L1-dcache-load-misses:u                                             (0.00%)
   <not supported>      LLC-loads:u
   <not supported>      LLC-load-misses:u

      0.015664218 seconds time elapsed

      0.000000000 seconds user
      0.003629000 seconds sys
```

Concurrent TCP client-server:



```
[root@archlinux c]# perf stat -d /usr/bin/taskset -c 0 ./server
Server is listening on port 8005
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Top CPU-consuming processes sent
Top CPU-consuming processes sent
Top CPU-consuming processes sent
Top CPU-consuming processes sent
^C/usr/bin/taskset: Interrupt

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

            18.78 msec task-clock                       #    0.001 CPUs utilized
                7      context-switches                 #  372.665 /sec
                1      cpu-migrations                   #   53.238 /sec
              261      page-faults                      #   13.895 K/sec
       69,681,847      cycles                           #    3.710 GHz                      (65.85%)
       16,923,803      stalled-cycles-frontend          #   24.29% frontend cycles idle     (68.06%)
       54,063,524      instructions                     #    0.78  insn per cycle
                                                  #    0.31  stalled cycles per insn     (68.06%)
       10,663,750      branches                         #  567.716 M/sec                    (63.28%)
          934,638      branch-misses                    #    8.76% of all branches          (79.25%)
       25,217,368      L1-dcache-loads                  #    1.343 G/sec                    (92.67%)
        1,347,148      L1-dcache-load-misses            #    5.34% of all L1-dcache accesses   (74.76%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

     25.188855556 seconds time elapsed

      0.002471000 seconds user
      0.016821000 seconds sys
```



```
[root@archlinux c]# perf stat -d /usr/bin/taskset -c 0 ./client 4
Thread 0: Hello message sent
Thread 1: Hello message sent
Thread 2: Hello message sent
Thread 3: Hello message sent
Thread 1: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 679300, System Time: 205310, Total Time: 884610
PID: 853, Name: kwin_wayland, User Time: 221580, System Time: 171690, Total Time: 393270

Thread 2: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 679300, System Time: 205310, Total Time: 884610
PID: 853, Name: kwin_wayland, User Time: 221580, System Time: 171690, Total Time: 393270

Thread 3: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 679300, System Time: 205310, Total Time: 884610
PID: 853, Name: kwin_wayland, User Time: 221580, System Time: 171690, Total Time: 393270

Thread 0: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 679300, System Time: 205310, Total Time: 884610
PID: 853, Name: kwin_wayland, User Time: 221580, System Time: 171690, Total Time: 393270


 Performance counter stats for '/usr/bin/taskset -c 0 ./client 4':

             1.64 msec task-clock                       #    0.084 CPUs utilized
                6      context-switches                 #    3.660 K/sec
                1      cpu-migrations                   #  609.962 /sec
              140      page-faults                      #   85.395 K/sec
        6,292,631      cycles                           #    3.838 GHz
        2,297,401      stalled-cycles-frontend          #   36.51% frontend cycles idle
        2,974,259      instructions                     #    0.47  insn per cycle
                                                  #    0.77  stalled cycles per insn
          684,501      branches                         #  417.520 M/sec
           85,794      branch-misses                    #   12.53% of all branches
     <not counted>      L1-dcache-loads                                               (0.00%)
     <not counted>      L1-dcache-load-misses                                         (0.00%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

      0.019486127 seconds time elapsed

      0.000000000 seconds user
      0.001797000 seconds sys
```

TCP client-server using “select”:

```
[root@archlinux c]# perf stat -d /usr/bin/taskset -c 0 ./server_select
Server is listening on port 8005
New connection, socket fd is 4
New connection, socket fd is 5
New connection, socket fd is 6
New connection, socket fd is 7
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Message received: Requesting CPU info from server
Top CPU-consuming processes sent
Top CPU-consuming processes sent
Top CPU-consuming processes sent
Top CPU-consuming processes sent
^C/usr/bin/taskset: Interrupt

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

            19.18 msec task-clock                #    0.003 CPUs utilized
                11      context-switches          #  573.432 /sec
                 1      cpu-migrations            #   52.130 /sec
               260      page-faults               #   13.554 K/sec
        76,411,993      cycles                    #    3.983 GHz                      (69.09%)
        18,950,188      stalled-cycles-frontend   #   24.80% frontend cycles idle     (68.72%)
        47,353,102      instructions              #    0.62  insn per cycle
                                                  #    0.40  stalled cycles per insn     (68.72%)
        11,062,867      branches                  #  576.709 M/sec                    (68.72%)
           916,553      branch-misses             #    8.28% of all branches          (68.72%)
        16,627,867      L1-dcache-loads           #  866.813 M/sec                    (80.98%)
           694,291      L1-dcache-load-misses     #    4.18% of all L1-dcache accesses   (81.94%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

       5.576206204 seconds time elapsed

       0.003893000 seconds user
       0.015563000 seconds sys
```

```
[root@archlinux c]# perf stat -d /usr/bin/taskset -c 0 ./client 4
Thread 0: Hello message sent
Thread 1: Hello message sent
Thread 2: Hello message sent
Thread 3: Hello message sent
Thread 0: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 695.06s, System Time: 210.01s, Total Time: 905.07s
PID: 853, Name: kwin_wayland, User Time: 227.66s, System Time: 175.98s, Total Time: 403.64s

Thread 3: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 695.06s, System Time: 210.01s, Total Time: 905.07s
PID: 853, Name: kwin_wayland, User Time: 227.66s, System Time: 175.99s, Total Time: 403.65s

Thread 2: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 695.06s, System Time: 210.01s, Total Time: 905.07s
PID: 853, Name: kwin_wayland, User Time: 227.66s, System Time: 175.99s, Total Time: 403.65s

Thread 1: Message received:
Top 2 CPU-consuming processes:
PID: 1611, Name: firefox, User Time: 695.06s, System Time: 210.01s, Total Time: 905.07s
PID: 853, Name: kwin_wayland, User Time: 227.66s, System Time: 175.98s, Total Time: 403.64s


 Performance counter stats for '/usr/bin/taskset -c 0 ./client 4':

             1.90 msec task-clock                #    0.093 CPUs utilized
                8      context-switches          #    4.212 K/sec
                1      cpu-migrations            #  526.548 /sec
              136      page-faults               #   71.610 K/sec
        4,431,468      cycles                    #    2.333 GHz                      (11.59%)
        2,379,947      stalled-cycles-frontend   #   53.71% frontend cycles idle
        2,978,209      instructions              #    0.67  insn per cycle
                                                 #    0.80  stalled cycles per insn
          685,820      branches                  #  361.117 M/sec
           85,779      branch-misses             #   12.51% of all branches
        1,298,890      L1-dcache-loads           #  683.928 M/sec                    (91.42%)
     <not counted>     L1-dcache-load-misses                                        (0.00%)
   <not supported>     LLC-loads
   <not supported>     LLC-load-misses

       0.020489449 seconds time elapsed

       0.001978000 seconds user
       0.000000000 seconds sys
```

Summary

This tabulated analysis contains only the server-side performance counters

Metric	Single-Threaded	Multithreaded without `select`	Multithreaded with `select`
Task Clock (msec)	7.43	18.78	19.18
CPU Utilization	0.000 CPUs (negligible)	0.001 CPUs	0.003 CPUs
Context Switches	3 (403.840/sec)	7 (372.665/sec)	11 (573.432/sec)
Page Faults	160 (21.538 K/sec)	261 (13.895 K/sec)	260 (13.554 K/sec)
Instructions per Cycle	0.69	0.78	0.62
Branch Misses	293,185 (9.11%)	934,638 (8.76%)	916,553 (8.28%)

- ***Task Clock (msec):*** The single-threaded implementation takes the least time because it handles fewer tasks concurrently, leading to lower overhead. The multithreaded implementations take more time as they involve more concurrency and overhead, which increases the time spent processing requests.
- ***CPU Utilization:*** CPU utilization is higher in multithreaded cases. The single-threaded version has almost negligible CPU usage because it handles requests one at a time. The version with select uses more CPU, which is expected since it's managing multiple file descriptors within a single thread, increasing overall load.
- ***Context Switches:*** The single-threaded implementation has very few context switches because there is only one active thread at a time. The multithreaded implementations show more context switches, with the select version showing the highest number because it involves additional work handling multiple connections in the same thread.
- ***Page Faults:*** The multithreaded versions show more page faults, which is expected because of the increased memory usage due to handling multiple clients simultaneously. However, both multithreaded versions show a similar number of page faults.
- ***IPC:*** In ‘select’, while the system is executing more instructions, it's doing so less efficiently due to increased overhead in managing multiple clients and context switches. The multithreaded without select has the highest IPC, which indicates more efficient execution compared to the select version.
- ***Branch misses:*** Lower misses in multithreading is likely due to better branch prediction as the CPU handles more predictable patterns when managing multiple threads or multiple clients.

Perf using VM:

Following image shows two linux running on VM, connected to each other, but since perf was not showing all the analytics, we preferred to do performance analysis comparison using native linux.