Network Benchmarking Report: Client-Server Socket Program

Shaurya Parashar 2022475   Shivankar Srijan 2022479

October 1, 2024

1 Introduction

This report details the performance analysis of various client-server models implemented using C socket programming. The experiments were conducted on ubuntu machine in separate network namespaces, and performance measurements were taken by pinning the processes to specific CPUs using `taskset`. Three types of server-client models were developed:

- Single-threaded server-client model
- Concurrent (multi-threaded) server-client model
- Select-based server-client model

The performance was evaluated using the `perf` tool to measure various performance counters like CPU cycles, instructions, cache misses, and context switches.

2 Socket Program Design

2.1 Server Design

The server was implemented with the following features:

1. Setup a TCP socket and listen for incoming client connections.
2. Handle multiple concurrent clients by creating a new thread for each client using the `pthread` library.
3. After accepting a connection, the server retrieves the top two CPU-consuming processes and sends the information to the client.
4. Use the `open()` system call to access the `/proc` filesystem and read `/proc/[pid]/stat` to gather process data.

2.2 Client Design

The client establishes a TCP connection to the server and sends a request to retrieve the CPU consumption details. Multiple clients can be initiated concurrently by passing the number of clients as a command-line argument.

2.3 Select-Based Model

The select-based server model uses the `select()` system call to handle multiple clients without multi-threading. This non-blocking approach allows the server to monitor multiple file descriptors (sockets) simultaneously, ensuring efficient resource utilization.

3 Implementation

The socket programs were written in C, and the performance was analyzed under various configurations:

- Single-threaded server-client
- Concurrent (multi-threaded) server-client
- Select-based server-client

The performance of these implementations was benchmarked using the `perf` tool on a Linux system with process pinning via `taskset`.

4 Testing Environment

The experiments were conducted on a Linux system where separate namespaces were created for isolating processes. The server and client programs were executed on different containers/VMs. The test involved analyzing the following metrics:

- CPU usage (cycles)
- Instruction count
- Context switches
- Elapsed time under different numbers of concurrent connections

5 Results

5.1 Cycle Count

The cycle count for the multithreaded and select-based server models was recorded across different numbers of clients:

Number of Clients	Multithreading (cycles)	Select (cycles)
3	51,001,065	62,316,952
6	127,096,626	126,676,492
10	204,929,848	208,161,972

Table 1: Cycle Count Comparison

Analysis:

- For 3 clients, the multithreaded approach is more efficient in terms of CPU cycles.
- With increasing client count (6 and 10 clients), both approaches use a similar number of cycles, indicating that both approaches scale similarly in terms of CPU usage when the client load increases.

5.2 Instruction Count

The number of instructions executed by the multithreaded and select-based server models was also measured:

Number of Clients	Multithreading (instructions)	Select (instructions)
3	41,808,593	49,803,028
6	100,622,940	98,624,574
10	167,345,487	164,571,162

Table 2: Instruction Count Comparison

Analysis:

- For 3 clients, multithreading uses fewer instructions.
- As the client count grows, the select-based server executes fewer instructions, suggesting that it remains more instruction-efficient at higher client counts.

5.3 Context Switches

The context switches for both server models were recorded:

Number of Clients	Multithreading (context switches)	Select (context switches)
3	4	9
6	26	18
10	50	31

Table 3: Context Switches Comparison

Analysis:

- Context switching is relatively low for both methods with 3 clients.
- Multithreading incurs significantly more context switches with 6 and 10 clients, whereas the select-based server performs better due to its single-threaded nature.

6 Conclusion

The performance analysis reveals that:

- The select-based server is more efficient for smaller client loads in terms of cycles, instructions, and elapsed time.
- The multithreaded approach incurs higher overhead initially but scales better at higher client counts, especially in terms of elapsed time.

Depending on the expected client load, one approach may be more suitable than the other.