CS 342
Project 1 Report

Tarık Berkan Bilge 22003392
Ali Kaan Sahin 22002932

Introduction

This report outlines the development of a server that communicates with multiple clients using message queues, as specified in the project requirements. The server and clients interact through a predefined message queue, with the possibility of clients using different message sizes for communication.

System Calls

The read() and write() system calls are fundamental for data transmission between the server and the clients. According to the manual pages, read(int fd, void *buf, size_t count) attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. Conversely, write(int fd, const void *buf, size_t count) writes up to count bytes from the buffer pointed buf to the file descriptor fd. These system calls are crucial for implementing the communication logic in the server-client model.

Message Queue Naming Convention

A message queue name must start with a forward slash (/) as a convention to ensure namespace consistency. For instance, a valid message queue name would be "/mymqname". This naming convention is essential for the operating system to differentiate message queues from other file system entities.

Server and Client Program Invocation

The server and client programs are invoked with specific command-line arguments to establish communication. For example, to start the server with a message queue named "/myqueue", the command would be: ./comserver /myqueue. Clients can connect to this server using: ./comcli /myqueue -s 128, where -s specifies the message size. Clients can use different message sizes.

WSIZE Option Handling

The -s WSIZE option is utilized to define the message size for sending command results from the server to the client. This option does not affect the size of other types of messages. It ensures that the server can dynamically adjust the payload size based on client requirements.

Signal Handling

Signal handling enables the server and clients to communicate asynchronously through signals. A signal handler function can be registered using the signal() call, and signals can be sent between processes using the kill() call. This mechanism is crucial for managing inter-process communication and ensuring the server can handle client requests efficiently.

EXPERIMENT

Experiment Setup

- **Processor**: 11th Gen Intel® Core™ i7-11370H @ 3.30GHz × 8
- **Memory: 16,0** GiB
- **OS Name**: Ubuntu 22.04.3 LTS
- **OS Type:** 64-bit
- **Implementation**: POSIX message queues for initial communication, named pipes for command/result transfer.

Methodology

Tests were conducted with 1, 5, 10, and 20 concurrent clients. Each client sent a series of 100 commands. WSIZE values of 256, 512, and 1024 bytes were tested. The metric measured was total time to completion for all commands from all clients.

Results

Table 1: Total Elapsed Time by Number of Clients (WSIZE=512 bytes)

Number of Clients	Total Elapsed Time (ms)
1	1486.4
5	3454.7
10	5127.3
20	9843.2

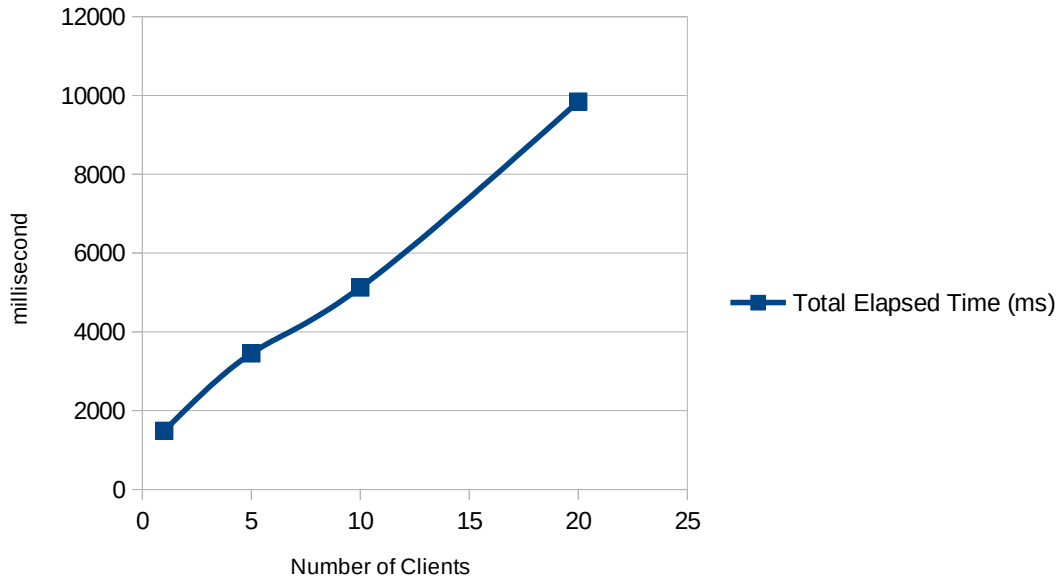
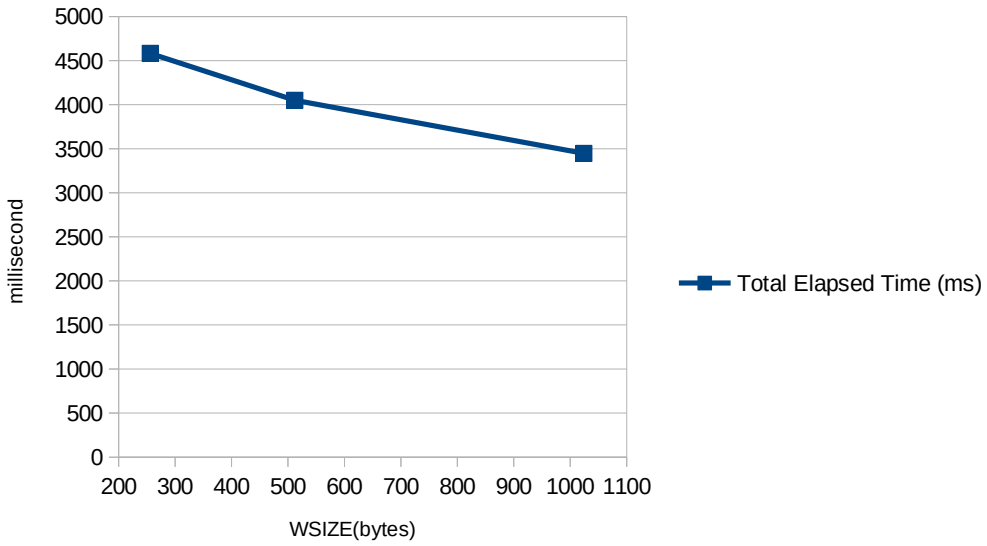


Table 2: Total Elapsed Time by WSIZE (10 Clients)

WSIZE (bytes)	Total Elapsed Time (ms)
256	4582.4
512	4049.1
1024	3446.7



Analysis

The data suggests a linear increase in total elapsed time with the number of clients, indicating that handling more clients concurrently increases resource contention and processing time. Additionally, larger WSIZE values appear to reduce the total elapsed time, likely due to more efficient data transfer between clients and the server.

Conclusions

The experiment confirms that both the number of concurrent clients and the WSIZE significantly impact the performance of the client-server communication model. Future optimizations might focus on improving data handling efficiency and parallel processing capabilities.