GTU CSE-344

Midterm

Md Sarwar Hossain

Id: 161044121

Client-Server Communication in C Programming

This report details a client-server communication program implemented in C, using FIFOs (named pipes) for inter-process communication, shared memory for process pool management, and a custom protocol for the communication between client and server.

Overview

The program consists of a server and multiple clients. The server and clients communicate through FIFOs. Each client sends a request to the server, the server processes the request and sends back a response. The client then reads the response and displays it to the user. The server also manages a pool of processes using shared memory.

The program includes several main files:

1. ProcessPool.h: This file defines the ProcessPool structure and associated operations. The process pool is stored in shared memory so it can be accessed by multiple processes. The operations include initializing the pool, adding a process, finding a process, and removing a process.
2. protocols.h: This file defines the communication protocols between the client and server. It includes request and response codes and a function to map string commands to request codes.
3. app.h: This file includes definitions and helper functions for the application, such as command parsing and validation.
4. client.c: This file implements the client-side logic. The client reads commands from the user, sends them to the server, and reads responses from the server.

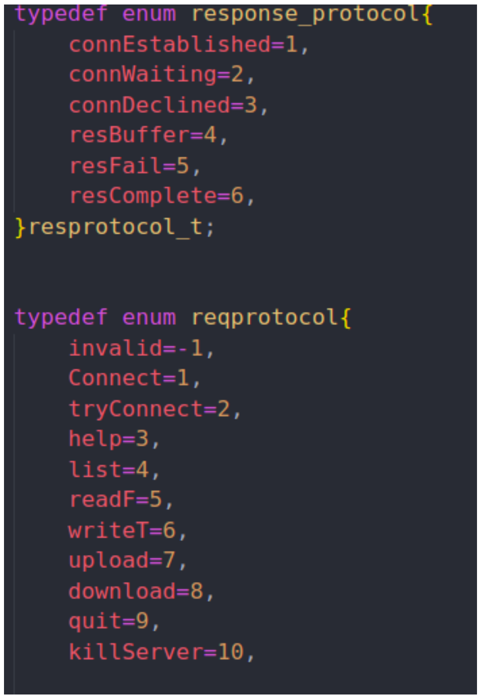
Design:

At first I have server fifo which is predetermined by template. I am creating a fifo with that one and waiting for clients to connect to that fifo. When a Client connects and send requests to the server , I check if they are already in processes queue. If the client is already in process queue than I go to server the client. I am holding clientpid to identify them. But if they are first time user, then I check if I reached the maximum connection limit, if not then I go and add the client in the process queue. After this first barrier, comes the part where I am serving the client.

Now comes the part how I am serving the clients. So clients also have a predefined template. After sending request to server , clients are waiting on the reading of the client FIFO. When server wants to server a client , at first Server forking the process. Then send response to to appropriate clientfifo throgh that child process. To Synchronise some of the client request I used two differenent mechanics.

I have used binary semaphores to sysnchronize add and remove to the processes queue. The reason to sysnchronize these two operation is that, they are in shared memory and any process can write. Using binary semaphores I made sure only one process can write at a time. Because of shared memory when a client signals killServer I can easily kill all the child processes as they are synchronized in the shared memory and accesible in real time by all process.

Client, part of the program was much easier. Once the client programs opens, at first it opens the clinet FIFO, then it opens server fifo and waits for server response. When the server response happens, it checks the type of response code. Based on the protocols the client understands what the server said.



```
typedef enum response_protocol{
    connEstablished=1,
    connWaiting=2,
    connDeclined=3,
    resBuffer=4,
    resFail=5,
    resComplete=6,
}resprotocol_t;


typedef enum reqprotocol{
    invalid=-1,
    Connect=1,
    tryConnect=2,
    help=3,
    list=4,
    readF=5,
    writeT=6,
    upload=7,
    download=8,
    quit=9,
    killServer=10,
```

I have also used some protocols when sending request to server, this ways the communication between server and client was much smoother. Once client gets resEstablished code it enters in a loop. A process in perspective of server. Then they message back and forth.

For Files, I used file locks to syncronize them. Once some file is being read, I put a read_lock with setkw that makes all other process that wants to write that file in block proceess mode. But other readers can read same time.

For write I put a write lock , so no one can read or write, when one process is writing.

Key Features

The client-server communication protocol supports various commands, including connection management (Connect, tryConnect), file operations (readF, writeT, list,help), server management (killServer), and others.

The client program (client.c) starts by validating command line arguments to ensure correct usage. It then creates a FIFO for communication with the server and sends a connection request to the server. The client reads the server's response and, if the connection is established, enters a loop where it reads commands from the user, sends them to the server, and reads the server's responses.

The client-side command parsing is robust and flexible, supporting commands with multiple arguments. It also handles signals such as SIGINT and SIGTERM gracefully, sending a quit command to the server before terminating.

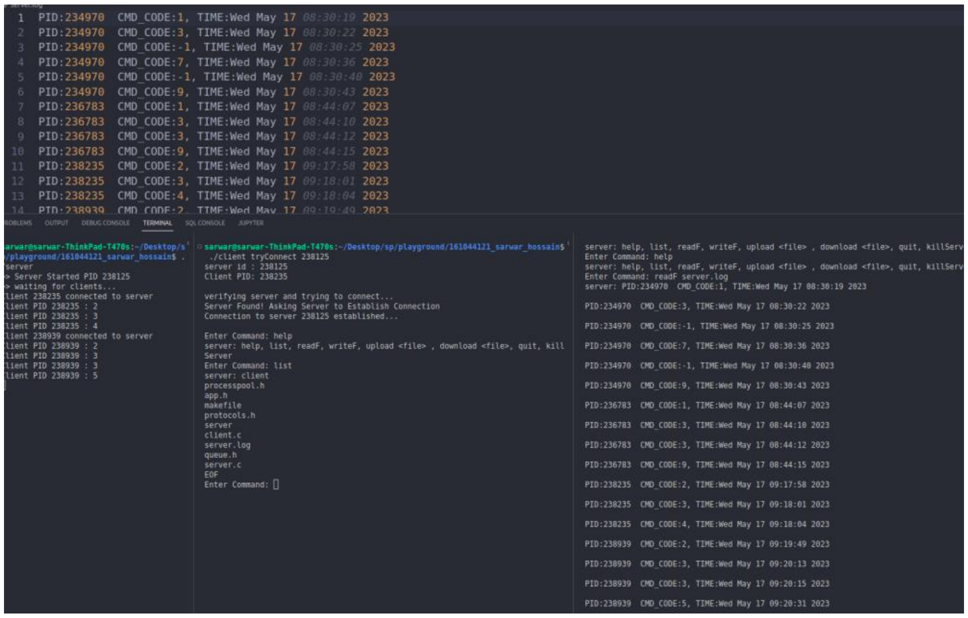
Conclusion

This program coveres inter-process communication, shared memory, custom communication protocols, and signal handling.

Drawbacks:

 I had the upload and download file implemented but at the last moment, I ran into a problem so I had to make the client and server static and in the same folder. Besides the signal handling could be more robust.

Running Program:



```
PID:234970  CMD_CODE:1, TIME:Wed May 17 08:30:19 2023
PID:234970  CMD_CODE:3, TIME:Wed May 17 08:30:22 2023
PID:234970  CMD_CODE:-1, TIME:Wed May 17 08:30:25 2023
PID:234970  CMD_CODE:7, TIME:Wed May 17 08:30:36 2023
PID:234970  CMD_CODE:-1, TIME:Wed May 17 08:30:40 2023
PID:234970  CMD_CODE:9, TIME:Wed May 17 08:30:43 2023
PID:236783  CMD_CODE:1, TIME:Wed May 17 08:44:07 2023
PID:236783  CMD_CODE:3, TIME:Wed May 17 08:44:10 2023
PID:236783  CMD_CODE:3, TIME:Wed May 17 08:44:12 2023
PID:236783  CMD_CODE:9, TIME:Wed May 17 08:44:15 2023
PID:238235  CMD_CODE:2, TIME:Wed May 17 09:17:58 2023
PID:238235  CMD_CODE:3, TIME:Wed May 17 09:18:01 2023
PID:238235  CMD_CODE:4, TIME:Wed May 17 09:18:04 2023
PID:238939  CMD_CODE:2, TIME:Wed May 17 09:19:49 2023
```

```
arwar@sarwar-ThinkPad-T470s:~/Desktop/s
/playground/161044121_sarwar_hossain$ .
/server
> Server Started PID 238125
> waiting for clients...
lient 238235 connected to server
lient PID 238235 : 2
lient PID 238235 : 3
lient PID 238235 : 4
lient 238939 connected to server
lient PID 238939 : 2
lient PID 238939 : 3
lient PID 238939 : 3
lient PID 238939 : 5
```

```
sarwar@sarwar-ThinkPad-T470s:~/Desktop/sp/playground/161044121_sarwar_hossain$
 ./client tryConnect 238125
server id : 238125
Client PID: 238235

verifying server and trying to connect...
Server Found! Asking Server to Establish Connection
Connection to server 238125 established...

Enter Command: help
server: help, list, readF, writeF, upload <file> , download <file>, quit, kill
Server
Enter Command: list
server: client
processpool.h
app.h
makefile
protocols.h
server
client.c
server.log
queue.h
server.c
EOF
Enter Command: 
```

```
server: help, list, readF, writeF, upload <file> , download <file>, quit, killServ
Enter Command: help
server: help, list, readF, writeF, upload <file> , download <file>, quit, killServ
Enter Command: readF server.log
server: PID:234970  CMD_CODE:1, TIME:Wed May 17 08:30:19 2023

PID:234970  CMD_CODE:3, TIME:Wed May 17 08:30:22 2023

PID:234970  CMD_CODE:-1, TIME:Wed May 17 08:30:25 2023

PID:234970  CMD_CODE:7, TIME:Wed May 17 08:30:36 2023

PID:234970  CMD_CODE:-1, TIME:Wed May 17 08:30:40 2023

PID:234970  CMD_CODE:9, TIME:Wed May 17 08:30:43 2023

PID:236783  CMD_CODE:1, TIME:Wed May 17 08:44:07 2023

PID:236783  CMD_CODE:3, TIME:Wed May 17 08:44:10 2023

PID:236783  CMD_CODE:3, TIME:Wed May 17 08:44:12 2023

PID:236783  CMD_CODE:9, TIME:Wed May 17 08:44:15 2023

PID:238235  CMD_CODE:2, TIME:Wed May 17 09:17:58 2023

PID:238235  CMD_CODE:3, TIME:Wed May 17 09:18:01 2023

PID:238235  CMD_CODE:4, TIME:Wed May 17 09:18:04 2023

PID:238939  CMD_CODE:2, TIME:Wed May 17 09:19:49 2023

PID:238939  CMD_CODE:3, TIME:Wed May 17 09:20:13 2023

PID:238939  CMD_CODE:3, TIME:Wed May 17 09:20:15 2023

PID:238939  CMD_CODE:5, TIME:Wed May 17 09:20:31 2023
```