Midterm Project

Student Name	Muhammed Oğuz
Student Number	1801042634

Table Of Contents

Problem Solution Approach

There was very much objectives for this project. I first start with main goals.

- Communicate between two process (FIFO)
- Communicate between parent and child with pipe
- Communicate between parent and child with shared memory

Communicate Between Two Processes

For communication between two different processes that are not relative each other. There is a communication method called `FIFO`. Another words, named pipes.

For handling and using those, there should be created a fifo file and use it between processes.

While communcation, it should be noted that, the other edge of file descriptor should be closed for proper results.

Pipe

For communication between parent and child, pipes are good alternative to shared memory.

They are very simillar like fifos.

Usages are the same.

Since, in this project, we are expected to handle more than 1 child processes. It should be noted that, all other edges of pipes should closed properly.

Shared Memory

This is the another way to handle communication between children and parent is shared memory.

For achieving this goal, it should be arranged a memory segment.

I use create and get functions for every shared memory segment for use create a better functionality.

It imeddiately creates or accesses shared memory and closes with returning the related void pointer.

Objectives and Solutions

0) Data

When creating the data, Those cases are considered as unvalid attempt due to implementation.

- When enter a matrix that is not square
- When entering a matrix that is n≤ 2.

1) Client X

Client sends a request with reading the data. After reading, forwards its matrix with fifo to server.

Server calculates the matrix and give response. While waiting the response, it waits on fifo read.

- Client can not run if server is not up.
- If client is down while sending data (with SIGINT). Server gets a blank request and says client is down.

2) Server Y

ServerY creates childs with given many times.

One of the biggest challange for serverY is determining which child (worker) available for handling the request.

I solved this with `shared memory`.

ServerY uses a shared memory, creates an array. And this array holds childs states and total counts of matrix info.

With this working system, ServerY, waits till a getting request from a client. Whenever gets a request, it searches and finds the next available child, If there is no available child during the time, it forwards request to serverZ.

Also, when serverY gets a `SIGINT`. It closes the reading pipe for exiting the waiting response state, after exiting the waiting response state, it immediately those the followings:

- Send `SIGINT` signal too all childerens, including ServerZ.
- Waits until all childrens are finished (prevent zombie)

- Close and unlink all shared memories (clean up shared memory segments)
- Remove serverYTemp file

`ServerYTemp` File

This file is used for prevent initializing more than once.

It created during starting the server, and removals from system on terminating the server.

If already a serverYTemp file appears, it prevents to initializing the server.

Also, client checks, if there is a serverYTemp file, it continues its execution, but if not, it says to starting the server.

3) Server Z

Server Z created during starting the server. It is a child of Server Y. It created by it.

After initializing, serverZ also created a child (workers) to wait response.

It uses shared memory.

It has a shared memory mechanisim to check which worker is available. (Apart from main shared memory mechanism)

It gets response from pipe of serverY and sends it with shared memory. This shared memory is handled by workers.

How ServerZ Workers Handled?

After send response to corresponding shared memory segment, ServerZ triggers a signal that called `SIGCONT`.

This signal allows to continue execution of a process if process is called with `SIGSTOP`.

Workers of ServerZ are works in while loop and in the beginning, they calls `SIGSTOP` for their own.

When they receive continue signal, they continues to evaluate their evalation.

4) Handling Deamon

I used the same mechanism from slides. But with BD_NO_CHDIR.

Sınce, I creating files and accessing files in same directory, I handle it with not changing the work directory.

Deamon works as expected. When checking controlling terminal, It also shows there is no controlling terminal.

Running and Testing

Testing

I wrote a very basic shell script.

```
#!/bin/bash
for ((i = 0; i < 20; i++))
do
  ./client -s serverFifo -o data/data.csv &
  sleep 1
  ./client -s serverFifo -o data/data2.csv &
  sleep 1
done
```

This sends clients with waiting in 1 seconds.

> 💡 Notice that, Server should be opened before running this script!

Makefile

Makefile has some good short commands to check every expectation of project.

Compile

With just typing `make` , it compiles both server and client

Running

`make run_server` runs the server. Command line arguments are changable from makefile

`make run_client` runs the client. Command line arguments are changable from makefile

Make Executable Test Script

`make test_with_script` command, changes the above script permission to executable to test properly.

Check Zombies

`make zombies` shows the output if there is a zombie process in system.

Check Shared Memory Leak

`make shared_mem_leak` shows the output if there is a memory leak.

It checks with `ipcs` and checks the `/dev/shm` location both.

Check Controlling Terminal

`make controling_terminal` checks the status of serverY,

It could be used for if there is a controling terminal or not.

Check Memory Leak

`make memory_client` runs client with valgrind.

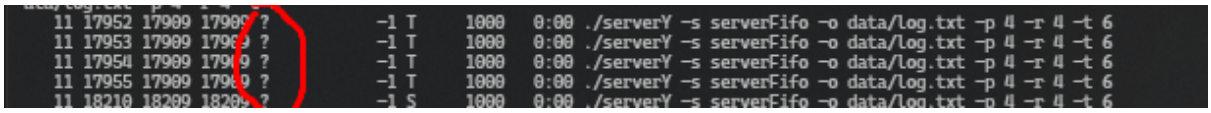
`make memory_server` runs server with valgrind.

Send SIGINT to Server

`make kill` this sends a SIGINT to server to finish.

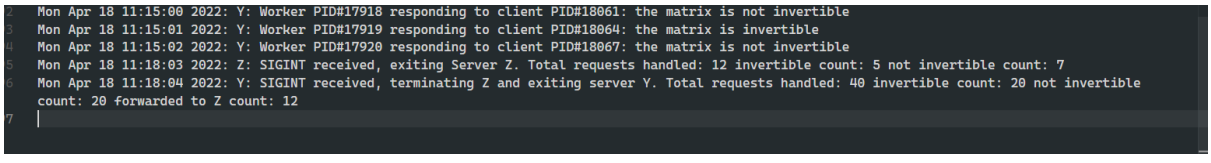
Results

No Controlling Terminal Result

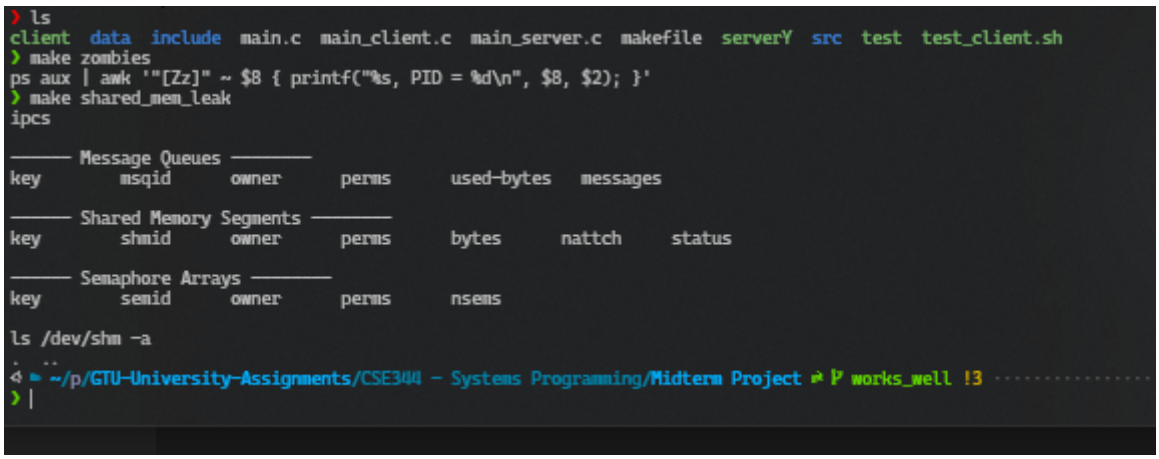


Why there is more than one instance? There should not be more than once instance.
The answer is, other results are the child processes of serverY.

Log File after Running Test

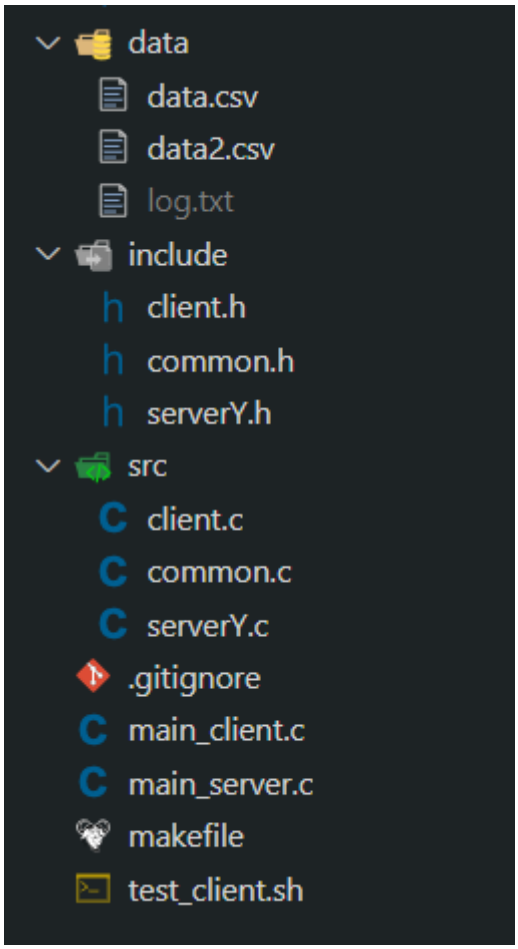


After execution Zombie Process - Shared Memory Results



Project Structure

Folder Structure



For better readibiltiy and maintabiltiy, I always divide my work to pieces.

Data Folder: This folder contains csv data. Also I chose to create log file in here.

Include Folder: This folder contains the header files

SRC Folder: This folder contains the implementations

Root Folder: This folder contains the main files and makefile.

Global Variables

I have some global variables to handle the project.

Most used global variable is `GLOBAL_ERROR` variable. this works like ERNO, but for me.

Also used some variables to handle with global signal handlers.

Less Comment More Functions

Due to clean code principles, I prefer to use less comments in my source code. I use meaningful variable names, divide functions to achieve readiblity.

Not Handled Parts

- Unfortunatelly, I use signal instead of sigcation. It may cause some troubles. But In my tests and my friends computer tests, It runs without problem.
- With very big client requests (e.g 2000), lower process counts (e.g 4) will not sufficient to handle the requests
- Like I wrote in the test script, If I don't give any sleep between requests, It may cause empty requests.
- Also, I used `SIGSTOP` and `SIGCONT`, But It would be nice I don't use those too. Regardless, I don't realized a downside of those signals.
- I do a lot of tests. I don't encountered any missing parts or errors expect those.