2023

System Programming HW4 Report

ELIFNUR KABALCI

1801042617

Project Steps:

(This part from midterm)

1) I wrote and edited the client-server codes on the 6th week slide from the materials used in the course. And I got it working. At this stage, first of all, I wanted to define a working code and server client connection.
   Compilation steps:
   gcc biboServer.c -o biboServer
   gcc biboClient.c -o biboClient

```
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboServer &
[1] 229
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboClient 3
0
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboClient 2
3
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboClient
5
```

2) I changed the code to be able to exchange messages between client and server. While running the server, I created the aforementioned direction using the sent direction and moved there. I equalized the other integer it took as a parameter to the maximum number of clients. I made fifo definitions for server and client files. For each process, I created a child with a fork and handled the request from the client. First of all, I defined the terminal commands specified in the assignment with if-else and tested it by just typing printf.

   In this part, I started to do my tests in two different terminals. First of all, I run the server area and wait for the client to connect. Then I connect the client from the client section and I made the assignments from the opened section.

```
latulipenoirez@Elifnur-PC:~/midterm1$ gcc biboServer.c -o biboServer
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboServer /home/latulipenoirez/midterm1 4
```

```
latulipenoirez@Elifnur-PC:~/midterm1$ gcc biboClient.c -o biboClient
latulipenoirez@Elifnur-PC:~/midterm1$ ./biboClient tryconnect 1
```

3) With the definitions made, I started to fill in the task fields of terminal commands. First of all, I implemented the easiest ones, help and list.

Server part outputs:

```
list
List of files:
biboClient
biboClient.c
get_num.h
biboServer.c
tlpi_hdr.h
biboServer.h
biboServer
error_functions.h
```

```
help
List of possible client requests:
- list: Display the list of files
- readF: Read contents of a file
- writeT: Write contents to a file
- upload: Upload a file to the server
- download: Download a file from the server
- help: Display the list of possible client requests
```

> In same time client part outputs:

```
Trying to connect to server (Server ID: 1)
>> Enter command: list
Server responded with seqNum 28
>> Enter command: help
Server responded with seqNum 28
```

4) I added log file and signal ls. Since I wrote these parts in the 2nd assignment, I copied them directly from there. However, I put it in the comment line because it gave an error at some test times.

Design:

I will tell you the design I made until the part where I can finish the homework. First of all, we open 2 sections in the terminal and apply the compilation lines that I finished in the project steps section. We open a server to one side and a client to the other. When the server is turned on, the client connection begins to wait. No matter how many clients you connect in the client section, it processes the maximum client amount. In the content of the assignment, I defined the request structs for the data sent from the client to the server, and the response structs for the data to be sent from the server to the client. In addition, definitions are made with the tlpi_hdr library in our starting code on the slide of the lesson. That's why I added the necessary ready-made libraries to the file.

Request struct contains pid seqlen and seq. These use seq to carry the message sent from the client to the server. It sends pid to show which client is processing. In the response part, I get the message directly by using the array named message.

The client data struct is defined to retrieve the data in the user entered from the client terminal screen. Finally, because I want to define shared memory, I defined the struct structure. However, I did not implement it because the similarity rate of the code I wrote with my research was low.

While the request is written and the response is read in the client part, the request is read and the response is written in the server part. This is what provides real communication. It is the fifo structure that lines up the clients.

HW4 additions:

First the progressions were what I did on the visa assignment. I did not make any changes in the client part. I set up an iterator structure to line up the header file a requests. I also used struct for this. I kept the pointer holding the struct and the following struct. As a first definition, I defined the head and tail parts of this iterator as null and made the first definition of mutex.

When I came to the main part of the server, I also took poolsize as an argument from the user. So I did the argument count check with 4, not 3. After I got the poolsize, I kept the thread pool definition in the thread array. I started the mutex. so that running threads will not interfere. I have defined threads. When I entered the loop, I deleted the parts other than reading the request from the server. I checked the mutex in lock and added the thread to the queue and handled it. After all the incoming data is gone, I deleted the threads. and I deleted the server data. Except for these interventions, the ones are the same as the visa.