

# **CSE-344 FINAL PROJECT**

**AHMET OKUR**

**1801042655**

## Problem Defination

The server side should be capable of handling multiple clients simultaneously, functioning as a multi-threaded internet server. Upon establishing a connection with the server, the directories on both the server and client sides need to be synchronized. This means that any new file created, deleted, or updated on the server should reflect the same changes on the client side, and vice versa.

## Plan & Design

### SERVER SIDE

- I set up the socket structure as **AF\_INET**. The port number was determined with the data received from the terminal.
- I created as many threads as the number of threads entered from the terminal. I designed the thread structure with the **producer-consumer** algorithm. I designated the producer as the main thread. The main thread accepts client connections and adds the received socket file descriptors to a queue.
- My consumer threads are used to read and process data from incoming client connections. The number of these worker threads is provided as one of the command-line arguments. For this synchronization, I used 1 mutex and 2 conditional variables.
- In my consumer thread, first the **listDir** method is called which scans the files and directories in the location where the server is located and adds this information to an array of type **struct fileInfo**. The **struct fileInfo** structure contains information such as the file name, path, access mode, size, and whether the file is a directory.
- File information is sent to the client by the **sendFileInfo** function. This function takes the information of a file in **struct fileInfo** format and sends this information to the client over a socket.
- The **receiveFileInfo** function, on the other hand, receives file information coming from the server and adds this information to an array of **struct fileInfo**. If the received file information specifies a directory, this directory is created on the client side.

- The array containing the file information from the server and the array containing the file information received from the client are sent to the **SaveDifferentElements** function. It is used to determine the differences between two different fileInfo arrays (here **serverFiles** and **receiveFiles**). This code takes into account a list that includes files with the same file paths and file sizes. After finding the differences, it saves these differences to another fileInfo array named **lastFiles**. Files that are present on the server but not from the client are saved to one array, and files that are present from the client but not on the server are saved to a different array.
- File reading and writing operations are performed with the **readFile** and **writeFile** functions. Files are read and written byte by byte over a socket. File sending and receiving operations are performed with the **sendFile** and **recvFile** functions.
- All of what I've written is within a loop and constantly checked. If there is a change in the client or server, the files get synchronized. After the first loop, the **removeFile** method comes into play. If a file or folder is deleted on the server, it sends this information to the client.

```

while (1)
{
    if (flag == 1)
    {
        countFile2 = 0;
        listDir(directory, 0, file2, &countFile2, strlen(directory));
        countRemovedFiles = SaveDifferentElements(files, file2, removedFiles, countServerFile, countFile2);
    }

    struct fileInfo receiveFiles[64];
    struct fileInfo sendFilesClient[64];
    struct fileInfo differencesFilesClient[64];
    countServerFile = 0;
    listDir(directory, 0, files, &countServerFile, strlen(directory));

    sendFileInfo(sock, files, countServerFile, removedFiles, countRemovedFiles);
    int countRecvFile = receiveFileInfo(sock, receiveFiles, directory);
    if (countRecvFile == -1)
    {
        write(1, "\nAny client logged out...\n", 27);
        break;
    }
    int countSendFilesCl = SaveDifferentElements(files, receiveFiles, sendFilesClient, countServerFile, countRecvFile);
    int countDiffFilesClient = SaveDifferentElements(receiveFiles, files, differencesFilesClient, countRecvFile, countServerFile);

    flag = 1;
    sendFile(sendFilesClient, countSendFilesCl, sock, directory);
    recvFile(differencesFilesClient, countDiffFilesClient, sock, directory);
}

```

THIS IS MY SERVER FILE SYNC OPERATIONS

## CLIENT SIDE

On the client side, I used the same methods as I used on the server side, the logic is the same.

```
logFile(NULL, NULL, 4, 0);
int i = 0;
while (1)
{
    struct fileInfo files[64];
    struct fileInfo receiveFiles[64];

    struct fileInfo sendFilesServer[64];
    struct fileInfo differencesFilesServer[64];

    int countClntFile = 0;

    int countRecvFile = receiveFileInfo(sock, receiveFiles,directory);
    listDir(directory, 0, files, &countClntFile, strlen(directory));
    sendFileInfo(sock, files, countClntFile,NULL,0);

    int countSendFilesServ = SaveDifferentElements(files, receiveFiles, sendFilesServer, countClntFile, countRecvFile);

    int countDiffFilesServ = SaveDifferentElements(receiveFiles, files, differencesFilesServer, countRecvFile, countClntFile);

    rcvFile(differencesFilesServer, countDiffFilesServ, sock, directory);
    if (countDiffFilesServ != 0)
        logging(differencesFilesServer, countDiffFilesServ, 2);

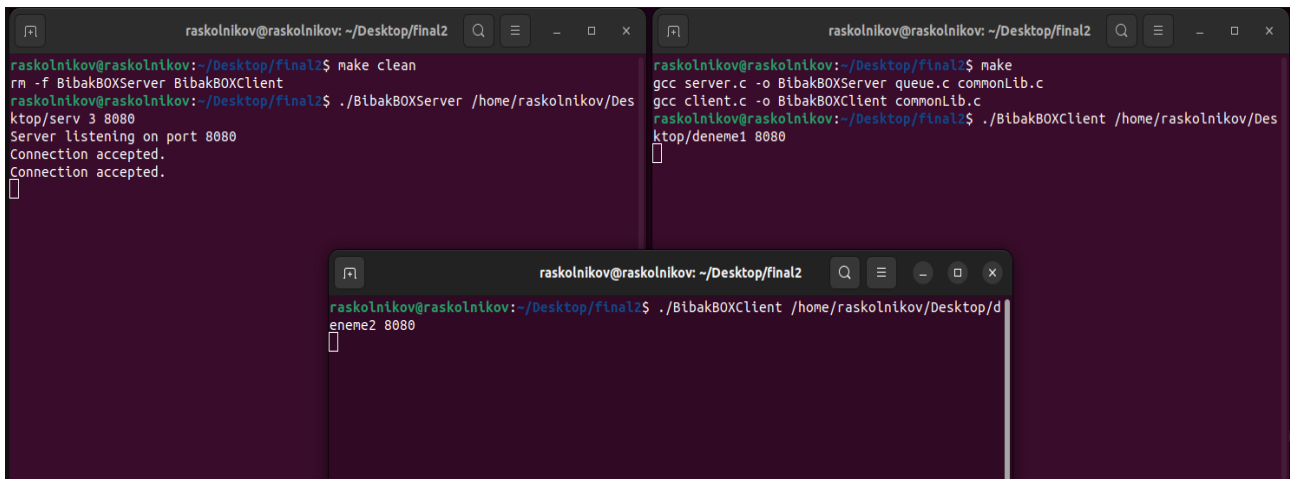
    sendFile(sendFilesServer, countSendFilesServ, sock, directory);
    if (countSendFilesServ != 0)
        logging(sendFilesServer, countSendFilesServ, 1);
}
```

A log file is kept on the client side.

## SIGNAL

In signal processing, I handled two signals, these are SIGINT and SIGPIPE. When SIGINT arrives at the client side, resources are released. Information is sent to the server side. When any side is closed, the SIGPIPE signal is sent to the other side, so I handled it.

# TEST CASES

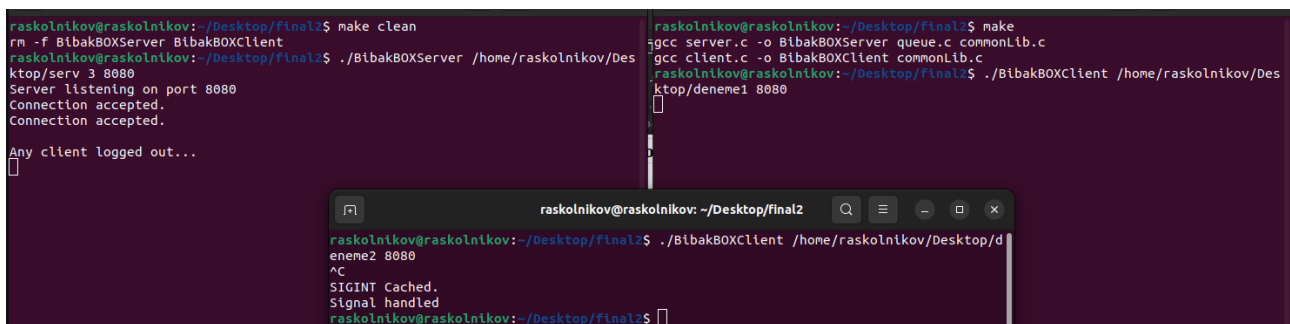


```
raskolnikov@raskolnikov: ~/Desktop/final2
raskolnikov@raskolnikov:~/Desktop/final2$ make clean
rm -f BibakBOXServer BibakBOXClient
raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXServer /home/raskolnikov/Desktop/deneme1 8080
ktp/serv 3 8080
Server listening on port 8080
Connection accepted.
Connection accepted.
[]

raskolnikov@raskolnikov:~/Desktop/final2$ make
gcc server.c -o BibakBOXServer queue.c commonLib.c
gcc client.c -o BibakBOXClient commonLib.c
raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXClient /home/raskolnikov/Desktop/deneme1 8080
[]

raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXClient /home/raskolnikov/Desktop/deneme2 8080
[]
```

This photograph is the image of a test with two clients; it was also tried with 10, 20, and 50 clients and it works. Here, the files have been synchronized.

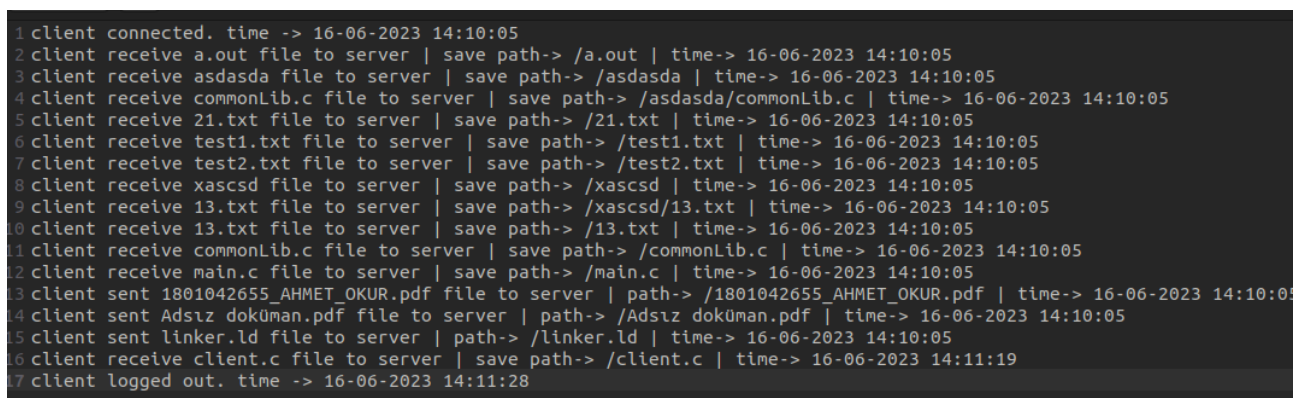


```
raskolnikov@raskolnikov:~/Desktop/final2$ make clean
rm -f BibakBOXServer BibakBOXClient
raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXServer /home/raskolnikov/Desktop/deneme1 8080
ktp/serv 3 8080
Server listening on port 8080
Connection accepted.
Connection accepted.
Any client logged out...
[]

raskolnikov@raskolnikov:~/Desktop/final2$ make
gcc server.c -o BibakBOXServer queue.c commonLib.c
gcc client.c -o BibakBOXClient commonLib.c
raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXClient /home/raskolnikov/Desktop/deneme1 8080
[]

raskolnikov@raskolnikov:~/Desktop/final2$ ./BibakBOXClient /home/raskolnikov/Desktop/deneme2 8080
AC
SIGINT Cached.
Signal handled
raskolnikov@raskolnikov:~/Desktop/final2$ []
```

Above, a SIGINT signal has been sent to one of the clients. The server has been informed.



```
1 client connected. time -> 16-06-2023 14:10:05
2 client receive a.out file to server | save path-> /a.out | time-> 16-06-2023 14:10:05
3 client receive asdasda file to server | save path-> /asdasda | time-> 16-06-2023 14:10:05
4 client receive commonLib.c file to server | save path-> /asdasda/commonLib.c | time-> 16-06-2023 14:10:05
5 client receive 21.txt file to server | save path-> /21.txt | time-> 16-06-2023 14:10:05
6 client receive test1.txt file to server | save path-> /test1.txt | time-> 16-06-2023 14:10:05
7 client receive test2.txt file to server | save path-> /test2.txt | time-> 16-06-2023 14:10:05
8 client receive xascsd file to server | save path-> /xascsd | time-> 16-06-2023 14:10:05
9 client receive 13.txt file to server | save path-> /xascsd/13.txt | time-> 16-06-2023 14:10:05
10 client receive 13.txt file to server | save path-> /13.txt | time-> 16-06-2023 14:10:05
11 client receive commonLib.c file to server | save path-> /commonLib.c | time-> 16-06-2023 14:10:05
12 client receive main.c file to server | save path-> /main.c | time-> 16-06-2023 14:10:05
13 client sent 1801042655_AHMET_OKUR.pdf file to server | path-> /1801042655_AHMET_OKUR.pdf | time-> 16-06-2023 14:10:05
14 client sent Adsız doküman.pdf file to server | path-> /Adsız doküman.pdf | time-> 16-06-2023 14:10:05
15 client sent linker.ld file to server | path-> /linker.ld | time-> 16-06-2023 14:10:05
16 client receive client.c file to server | save path-> /client.c | time-> 16-06-2023 14:11:19
17 client logged out. time -> 16-06-2023 14:11:28
```

The above is the log file.