CSE 344 HOMEWORK 4
MYBIBOBOX
A CONCURRENT FILE ACCESS SYSTEM

Çağrı Çaycı
1901042629

OVERVIEW

The purpose of this homework, providing server-client communication in multithreading manner. To accomplish this purpose, some concepts such as inter-process communication, POSIX threads, file input-output, busy waiting, signal handlers must be used. Listing the files in the server directory, reading a file, writing to a file, upload a file to server, download a file from the server, and killing the server are some of the requests between client and server. After all the concepts work together in harmony, these requests are handled by server properly and the server returns the appropriate response to the client.

CONCEPTS

INTER-PROCESS COMMUNICATION

One of the first concepts to be applied when designing a server-client system is inter-process communication. Server and client must communicate each other to transfer requests and responses. In this project, FIFO is used to provide inter-process communication. The reason for using FIFO is avoiding synchronization issues of the shared memory. Three FIFOs are used in this project. First of the FIFOs is server FIFO which server reads it constantly for new clients. Second of the FIFOs is clientRead FIFO which sends the requests of the client to the server. The last FIFO is clientWrite FIFO which sends the response of the server to client. The reason for using two different FIFO to write and to read is avoiding synchronization issues and data corruption

```
while(strncmp(buffer, "quit", 4) != 0){

    printf(">>");

    fgets(buffer, sizeof(buffer), stdin);

    if(strncmp(buffer, "upload", 6) == 0 || strncmp(buffer, "download", 8) == 0){
        char cwd[256];
        getcwd(cwd, sizeof(cwd));
        buffer[strlen(buffer) - 1] = '\0';
        sprintf(task, "%d %s %s", getpid(), buffer, cwd);
    }else
        sprintf(task, "%d %s", getpid(), buffer);

    write(clientWrite, task, sizeof(task));

    memset(response, 0, sizeof(response));

    read(clientRead, response, sizeof(response));

    printf("%s", response);
};
```

(Client writes to clientWrite FIFO and reads the clientRead FIFO.)

```
while(1){
    memset(FIFO, 0, sizeof(FIFO));

    if(read(server_fifo, FIFO, sizeof(FIFO)) > 0){ /* Read the server fifo. */

        int client_pid = atoi(strtok(FIFO, " "));
```

(Server reads the server FIFO for new clients)

```
while(1){
    memset(buffer, 0, sizeof(buffer));

    if(read(client_fifo, buffer, sizeof(buffer)) > 0){
        Task task = {.task_func = &task_func};
```

(Server reads the clientWrite FIFO for requests of client)

POSIX THREADS

In this project, thread pool is created for each client in the server. When a client request something from the server, a task is created according to the request, and one of the threads in the thread pool handles the request. The size of the thread pool is got from the user as argument. Some of the components such as mutexes and condition variables are used to provide synchronization and decrease CPU usage in computer.

```
pthread_t threadPool[number_of_threads];
pthread_mutex_init(&mutex, NULL);
for(int i = 0; i < number_of_threads; i++){
    if(pthread_create(&threadPool[i], NULL, &start, buffer) != 0){
        errExit("phread_create");
    }
}

while(1){
    memset(buffer, 0, sizeof(buffer));

    if(read(client_fifo, buffer, sizeof(buffer)) > 0){
        Task task = {.task_func = &task_func};
        task.request = (char *) malloc(sizeof(buffer));
        strcpy(task.request, buffer);
        add(task);
    }
}
pthread_mutex_destroy(&mutex);
```

(Thread pool is created. Mutex and each thread in the thread pool is initialized. After the execution mutex is destroyed.)

```
void add(Task task){
    pthread_mutex_lock(&mutex);
    taskQue[number_of_tasks] = task;
    number_of_tasks++;
    pthread_mutex_unlock(&mutex);
    pthread_cond_signal(&cond);
}

void * start(){
    while(1){
        pthread_mutex_lock(&mutex);

        while(number_of_tasks == 0){
            pthread_cond_wait(&cond, &mutex);
        }

        Task task = taskQue[0];

        for(int i = 0; i < number_of_tasks - 1; i++){
            taskQue[i] = taskQue[i + 1];
        }
        number_of_tasks -= 1;

        pthread_mutex_unlock(&mutex);

        task.task_func(task.request);
    }
}
```

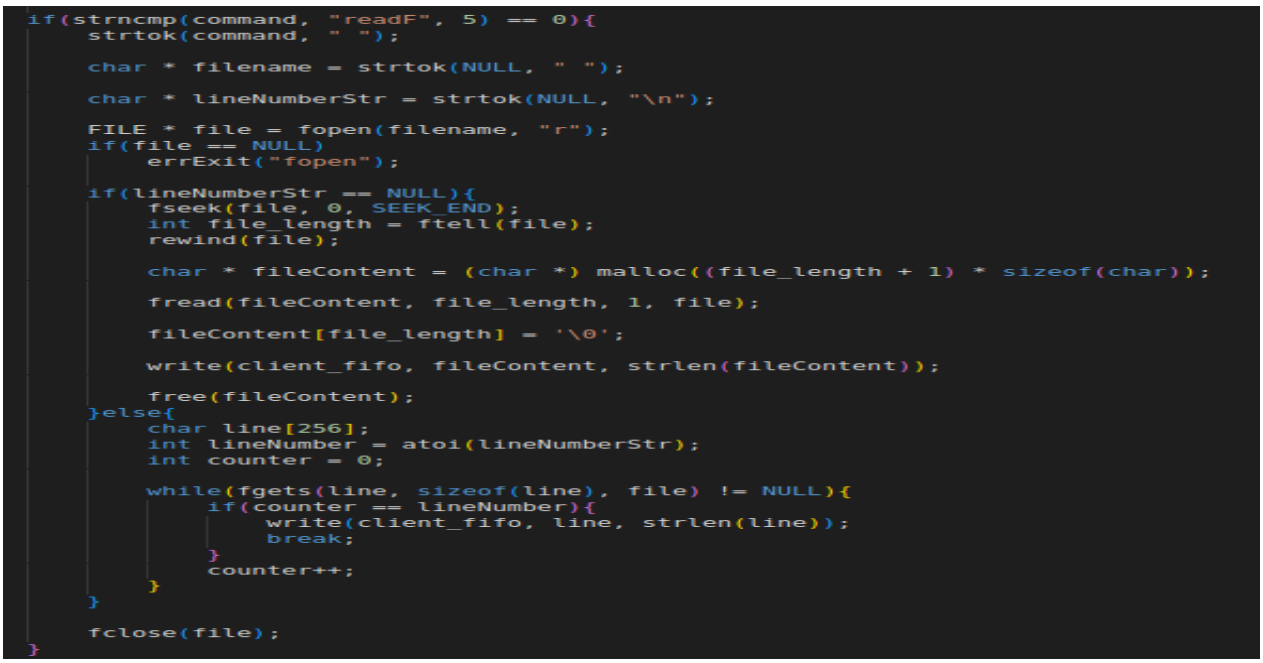
(add function adds a task to taskQue. The operation occurs in critical region to avoid interrupts and sends signal to condition variable after its done.)

(start function gets first task from the taskQue and executes it. It waits until there is a task in the taskQue.)

(task_func function handles the task according to its argument.)

FILE INPUT-OUTPUT

In this project, most of the requests of the clients contain file input-output operations. To return appropriate response to the client, file operations must be handled correctly.



```
if(strncmp(command, "readF", 5) == 0){
    strtok(command, " ");

    char * filename = strtok(NULL, " ");

    char * lineNumberStr = strtok(NULL, "\n");

    FILE * file = fopen(filename, "r");
    if(file == NULL)
        errExit("fopen");

    if(lineNumberStr == NULL){
        fseek(file, 0, SEEK_END);
        int file_length = ftell(file);
        rewind(file);

        char * fileContent = (char *) malloc((file_length + 1) * sizeof(char));

        fread(fileContent, file_length, 1, file);

        fileContent[file_length] = '\0';

        write(client_fifo, fileContent, strlen(fileContent));

        free(fileContent);
    }else{
        char line[256];
        int lineNumber = atoi(lineNumberStr);
        int counter = 0;

        while(fgets(line, sizeof(line), file) != NULL){
            if(counter == lineNumber){
                write(client_fifo, line, strlen(line));
                break;
            }
            counter++;
        }
    }

    fclose(file);
}
```

(Reading file operation: If the command is “read all file”, allocates the memory according to size of the file and stores the file content in it. Otherwise, reads the file content line by line, when the correct line is found, stores it.)

```
else if(strncmp(command, "writeT", 6) == 0){
    strtok(command, " ");

    char * filename = strtok(NULL, " ");

    char * lineNumberStr = strtok(NULL, " ");

    char * string = strtok(NULL, "\n");

    int lineNumber = atoi(lineNumberStr);

    if(lineNumber == 0 && strcmp(lineNumberStr, "0") != 0){
        FILE * file = fopen(filename, "a+");
        if(file == NULL)
            errExit("fopen");

        fprintf(file, "%s %s", lineNumberStr, string);
        fclose(file);
    }else{
        char line[256];
        int current = 0, position = 0;

        FILE * file = fopen(filename, "r");
        if(file == NULL)
            errExit("fopen");

        FILE * tmp = fopen("temp.txt", "w");
        if(tmp == NULL)
            errExit("tmpfile");

        while(fgets(line, sizeof(line), file) != NULL){
            if(current != lineNumber){
                fputs(line, tmp);
            }
            else{
                fprintf(tmp, "%s\n", string);
            }
            current++;
        }
        fclose(file);
        fclose(tmp);

        remove(filename);
        rename("temp.txt", filename);

    }
```

(Writing to file operation: If the line number is not given, appends the string to at the end of the file. Otherwise, creates a temp file and writes contents of the file line by line to temp file and replaces the given line with string. At the end removes the old file and renames the temp file.)

```
else if(strncmp(command, "upload", 4) == 0){
    strtok(command, " ");

    char * filename = strtok(NULL, " ");

    char * path = strtok(NULL, "\n");

    char exactPath[124], line[256];

    sprintf(exactPath, "%s/%s", path, filename);

    FILE * fileOriginal = fopen(exactPath, "r");

    FILE * fileNew = fopen(filename, "w");

    while(fgets(line, sizeof(line), fileOriginal) != NULL){
        fputs(line, fileNew);
    }

    fclose(fileOriginal);

    fclose(fileNew);

    strcpy(response, "Task is Completed!\n");

    write(client_fifo, response, sizeof(response));
```

(Upload a file: Opens the original file in read mode and opens a new file in the directory of the server. Reads the original file line by line and writes to new file.)

```
}else if(strncmp(command, "download", 8) == 0){
    strtok(command, " ");

    char * filename = strtok(NULL, " ");

    char * path = strtok(NULL, "\n");

    char exactPath[124], line[256];

    sprintf(exactPath, "%s/%s", path, filename);

    FILE * fileOriginal = fopen(filename, "r");

    FILE * fileNew = fopen(exactPath, "w");

    while(fgets(line, sizeof(line), fileOriginal) != NULL){
        fputs(line, fileNew);
    }

    fclose(fileOriginal);

    fclose(fileNew);

    strcpy(response, "Task is Completed!\n");

    write(client_fifo, response, sizeof(response));
```

(Download a file: Opens the original file in read mode and opens a new file in the directory of the client. Reads the original file line by line and writes to new file.)

```
 else if(strncmp(command, "list", 4) == 0){
    DIR * direct = opendir(".");
    struct dirent * entry;
    char files[1024];
    if(direct == NULL)
        errExit("opendir");

    while((entry = readdir(direct)) != NULL){
        strcat(files, entry->d_name);
        strcat(files, "\n");
    }
    write(client_fifo, files, sizeof(files));

    closedir(direct);
}else{
    strcpy(response, "Command not written properly.\n");

    write(client_fifo, response, sizeof(response));
}
```

(List the files in directory of the server: Opens the directory of the server and gets file names one by one. Stores them in a string.)

BUSY WAITING

In this project, the maximum number of clients is got from the user when the server is started as an argument. If the number of clients in the server is equal to maximum number of clients, new client (assume it is started with connect argument) must wait until a spot is available in the server.

```
for(int i = 0;; i++){ /* Find available spot for the client. */
    int position = i % number_of_clients;
    if(kill(clients[position], 0) != 0)
        clients[position] = 0;

    if(clients[position] == 0){
        clients[position] = client_pid;
        current = position;
        curClients++;
        break;
    }

    if(i == number_of_clients - 1){
        printf(">>Client PID %d cannot be connected to server. Que is FULL!\n", client_pid);
        fflush(stdout);
        if(strncmp(connect, "connect", 7) != 0){
            kill(client_pid, SIGTERM);
            break;
        }
    }

    if(position == number_of_clients)
        sleep(1);
}

if(clients[current] != client_pid) /* Check whether client can connected to the server. */
    continue;

printf(">>Client PID %d is connected to server as client%d.\n", client_pid, curClients);
```

(All the clients are checked whether they are running with kill function. If a spot is available, client is placed there and breaks the loop. If the client tries to connect with an argument different than “connect” and there are no available spot, it is terminated.)

HANDLING SIGNALS

Client can be terminated with either CRTL-C or client request. To terminate the client properly with CTRL-C, SIGTERM signal must be handled.

```
void SIGINTHandler(int signal){
    char buffer[100];
    sprintf(buffer, "%d quit", getpid());
    write(clientWrite, buffer, sizeof(buffer));
    sleep(1);
    close(clientWrite);
    close(clientRead);
    exit(0);
}
```

(Sends quit request to server and close the FIFOs properly.)

WORKING PRINCIPLE

The working principle of the project is like midterm project. Firstly, server creates a FIFO and continuously read the FIFO. Every client can connect to this FIFO with server PID. When a client connects to FIFO, client sends its own PID to FIFO. So, server can connect to FIFO of the client with this PID. Server creates a child, and this child continuously reads client FIFO to get requests of client. The thread which is got from the thread pool handles client request and send response to client by using client FIFO. In this way, every child connected to the server FIFO can communicate with server continuously.

TESTS

READING A FILE

```
cagri@CSE344:~/MasaüstÜ/HW4/client$ cat cagri.txt
cat: cagri.txt: No such file or directory
cagri@CSE344:~/MasaüstÜ/HW4/client$ cd ..
cagri@CSE344:~/MasaüstÜ/HW4$ cat cagri.txt
Hello
my
name
is
cagri
1901042629
Gebze Technical University Computer Engineering Department
cagri@CSE344:~/MasaüstÜ/HW4$ cd client
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4117
>>readF cagri.txt
Hello
my
name
is
cagri
1901042629
Gebze Technical University Computer Engineering Department
>>readF cagri.txt 0
Hello
>>readF cagri.txt 1
my
>>readF cagri.txt 5
1901042629
>>readF cagri.txt 6
Gebze Technical University Computer Engineering Department
>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$
```

WRITING TO FILE

```
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4117
>>readF cagri.txt
Hello
my
name
is
cagri
1901042629
Gebze Technical University Computer Engineering Department
>>writeT cagri.txt i am adding these string
Task is Completed!
>>readF cagri.txt
Hello
my
name
is
cagri
1901042629
Gebze Technical University Computer Engineering Department
i am adding these string>>writeT cagri.txt 3 this is middle
Task is Completed!
>>readF cagri.txt
Hello
my
name
this is middle
cagri
1901042629
Gebze Technical University Computer Engineering Department
i am adding these string>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$
```

LISTING THE FILES

```
cagri@CSE344:~/MasaüstÜ/HW4$ ls
biboServer  cagri.txt  client  log  Makefile  server.c
cagri@CSE344:~/MasaüstÜ/HW4$ cd client
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4117
>>list
..
server.c
biboServer
Makefile
cagri.txt
.
client
log
.vscode
>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$
```

HELP

```
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4117
>>help
Available commands are:
help, list, readF, writeT, upload, download, quit, killServer
>>help list
list
        display the list of files in servers directory
>>help readF
readF <file> <line #>
        display the #th line of the <file>, returns with an error if <file> does not exist
>>help writeT
writeT <file> <line #> <string>
        request to write the content of "string" to the #th line the <file>, if the line # is not given writes to the end of file. If the file does not exists in Servers directory creates and edits the f
le at the same time
>>help upload
upload <file>
        upload the file from the current working directory of client to the servers directory
>>help download
download <file>
        request to receive <file> from servers directory to client side
>>help quit
quit
        quit
>>help killServer
killServer
        kill the server
>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$
```

UPLOADING A FILE

```
cagri@CSE344:~/MasaüstÜ/HW4$ ls
biboServer  client  log  Makefile  server.c
cagri@CSE344:~/MasaüstÜ/HW4$ cd client
cagri@CSE344:~/MasaüstÜ/HW4/client$ ls
biboClient  cagri.txt  client.c
cagri@CSE344:~/MasaüstÜ/HW4/client$ cat cagri.txt
Hello
my
name
is
Cagri
Gebze Technical University Computer Engineering
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4950
>>upload cagri.txt
Task is Completed!
>>readF cagri.txt
Hello
my
name
is
Cagri
Gebze Technical University Computer Engineering
>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$
```

DOWNLOADING A FILE

```
cagri@CSE344:~/MasaüstÜ/HW4/client$ ls
biboClient  client.c
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 4117
>>download cagri.txt
Task is Completed!
>>quit
Sending write request to server log file.
Waiting for log file
Log file write request granted
bye...
cagri@CSE344:~/MasaüstÜ/HW4/client$ ls
biboClient  cagri.txt  client.c
cagri@CSE344:~/MasaüstÜ/HW4/client$ cat cagri.txt
Hello
my
name
this is middle
cagri
1901042629
Gebze Technical University Computer Engineering Department
i am adding these stringcagri@CSE344:~/MasaüstÜ/HW4/client$
```

KILLING THE SERVER

```
cagri@CSE344:~/MasaüstÜ/HW4/client$ ./biboClient connect 5030
>>help
Available commands are:
help, list, readF, writeT, upload, download, quit, killServer
>>list
..
server.c
biboServer
Makefile
cagri.txt
.
client
log
.vscode
>>killServer
```

```
cagri@CSE344:~/MasaüstÜ/HW4$ make
gcc server.c -o biboServer
gcc client/client.c -o client/biboClient
cagri@CSE344:~/MasaüstÜ/HW4$ ./biboServer log 3 5
>>Server started with PID 5030
>>Waiting for clients...
>>Client PID 5031 is connected to server as client1.
>>Kill signal from client1...Terminating...
>>Bye...
Killed
cagri@CSE344:~/MasaüstÜ/HW4$
```