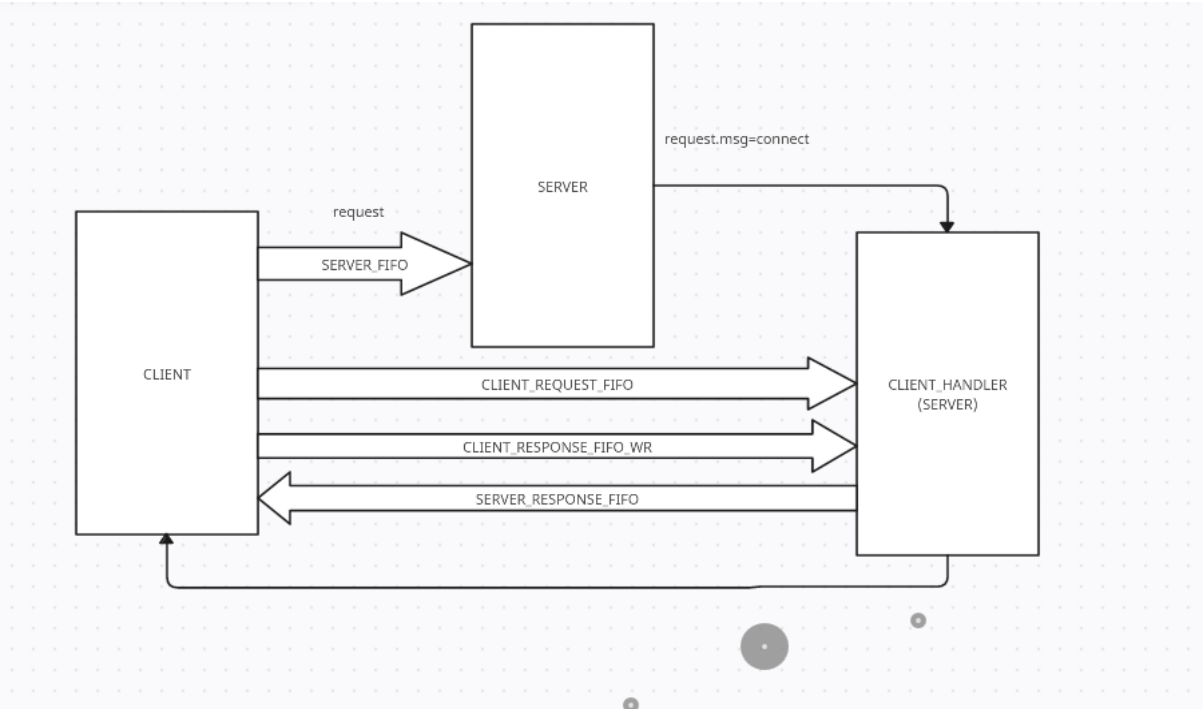
CSE 344 SYSTEM PROGRAMMING 2022/2023

MIDTERM REPORT

Muhammed Sİnan Pehlivanoğlu

1901042664

SYSTEM ARCHITECTURE



DESIGN CHOICE

- Accepting Connection Request

To connect server, Client sends Connection request using the write end of SERVER_FIFO fifo. Server creates SERVER_FIFO as reader mode to accept request from user before client wants connection.

- Sending Specific Commands to Server

After the connection is ensured by the Server, Client creates CLIENT_REQUEST_FIFO. Also Server forks a proccess to be responsible for Client. Clients use write end of the CLIENT_REQUEST_FIFO to send command request to Server. Server listens the CLIENT_REQUEST_FIFO.

- Sending Response Data to Client from Server

After the connection is ensured by the Server, Client also creates CLIENT_RESPONSE_FIFO. Client listens CLIENT_RESPONSE_FIFO to get response from Server. Server uses write end of the CLIENT_RESPONSE_FIFO.

- Sending Response Data to Server from Client

The Server may wants necessary data from Client. So CLIENT_RESPONSE_WR is used to ensure that data transfer between Client and Server. Server listens CLIENT_RESPONSE_WR and client writes response to CLIENT_RESPONSE_WR .

****

APPROACHES TO POSSIBLE SYNCHRONIZATION PROBLEMS

- If the operation is doing on file then only one client has permission to read or write this file. File locking mechanism is used for handling this race conditions.
- There is a waiting queue that stores client waiting. It is shared memory.

```
waiting_Que_shmid = shm_open("/queue_shm", O_CREAT | O_RDWR, 0644);
if(waiting_Que_shmid == -1){
    perror("shm_open : ");
    exit(EXIT_FAILURE);
}
ftruncate(waiting_Que_shmid, sizeof(Queue));
waiting_QUE = (Queue*) mmap(NULL, sizeof(Queue), PROT_READ | PROT_WRITE, MAP_SHARED, waiting_Que_shmid, 0);
if (waiting_QUE == MAP_FAILED) {
    perror("mmap");
    exit(EXIT_FAILURE);
}
waiting_QUE->rear = 99;
waiting_QUE->front = 99;
waiting_QUE->size = 100;

waiting_QUE->array = (long int *) mmap(NULL, sizeof(int) * 100, PROT_READ | PROT_WRITE, MAP_SHARED, waiting_Que_shmid, 0);
```

- If the request is “Connect” , client is added to queue. IF the request is “tryConnect” client is added to queue if there is a enough space.

```
    if(strcmp(request.msg, "Connect") == 0){

        enqueue(waiting_QUE,request.pid);
    }
    else if(strcmp(request.msg, "tryConnect") == 0){

        if(*client_num_shm < max_clients){
            enqueue(waiting_QUE,request.pid);
        }
        else{
            kill(request.pid, SIGTERM);
        }
    }
```

- Number of client number is shared memory that is value is changed in child proccess.

```
client_num_shm = mmap(NULL, sizeof(int), PROT_READ \
                 | PROT_WRITE, MAP_SHARED, client_num_shm_fd, 0);

if(client_num_shm == MAP_FAILED){
    perror("mmap failed : ");
    exit(EXIT_FAILURE);
}
```

****

- **Handling Lİst Command**

```c
int handle_list(int fd){
    int retval = 0;
    pid_t pid = fork();
    if(pid < 0){
        perror("Error : ");
        retval = -1;
    }
    if(pid == 0){
        dup2(fd, STDOUT_FILENO);
        execl("/bin/ls", "ls","-l","server_files" ,NULL);
        perror("execl");
        retval = -1;
    }
    else{
        int status;
        waitpid(pid, &status, 0);
        if (WIFEXITED(status)) {
            //printf("Child exited with status %d\n", WEXITSTATUS(status));
        } else {
            perror("Error : ");
            retval = -1;
        }
    }
    close(fd);
    return retval;

}
```

List command use ls command. So we need to first create child process . Also we need to use STDOUT as duplication of Client_Response_fd. Then the image of child process is changed with execl system call. So the output of the ls command now redirected to Client_Response_fd.

- **Handling Help Command**

```c
int handleHelp(const Request request, Response response, const int client_res_fd){
    int retval = 0;
        if(strcmp(request.msg , "help" ) == 0){

            strncpy(response.package, "\tAvaliable Comments are:\nhelp, list, readF, writeT, upload, download, quit, killServer\n",sizeof(Response));

        }
```

. . .

```c
        if(write_to_file(client_res_fd, sizeof(response), &response ) == -1){
            retval = -1;

        }
return retval;
```

The desired help command result is writed to Client_Response_Fifo

- **Handling ReadF in Server Side**

```c
int handle_readF(Request request, Response response, const int client_res_fd){
    char* words[2];
    int i;
    int word_count;
    int retval = 0;
    int fd;
    char file_name[256];
    // SPLITTING REQUEST
    for(i = 0 ; i < 2; ++i ){
        words[i] = (char *) malloc(sizeof(char)* 256);
    }
    word_count = split_string_into(request.msg, words,2);
    if( word_count == -1){
        retval = -1;
        for(i = 0 ; i < 2; ++i ){
            free(words[i]);
        }
    }
    else{
        // CHECK IS FILE EXIST
        snprintf(file_name,256,"server_files/%s",words[1]);
        if(access(file_name, F_OK) == -1){
            memset(&response,'\0', sizeof(Response));
            strncpy(response.package, "Error Code", 33);
            write_to_file(client_res_fd, sizeof(Response), &response);
            retval = -1;
        }
        else{
            //OPEN DESIRED FILE
            fd = open(file_name, O_RDONLY);
            if(fd == -1){
                memset(&response,'\0', sizeof(Response));
                strncpy(response.package, "Error Code", 33);
                retval = -1;
            }
```

```c
        else{
            //OPEN DESIRED FILE
            fd = open(file_name, O_RDONLY);
            if(fd == -1){
                memset(&response,'\0', sizeof(Response));
                strncpy(response.package, "Error Code", 33);
                retval = -1;
            }
            else{
                // LOCK FILE TO ENSURE SYNC
                if(flock(fd , LOCK_SH) == -1){
                    memset(&response,'\0', sizeof(Response));
                    strncpy(response.package, "Error Code", 33);
                    retval = -1;
                }
                else{
                    memset(&response,'\0', sizeof(Response));
                    strncpy(response.package, "Found", 33);
                }
                // WRITE TO THE CLIENT_RESPONSE_FIFO
                write_to_file(client_res_fd, sizeof(Response), &response);
                if((retval != -1) && read_file_into(fd, &response, sizeof(Response), client_res_fd) == -1){
                    retval = -1;
                }
                // UNLOCK FILE
                if(flock(fd, LOCK_UN) == -1){
                    perror("Error unlocking file");
                    retval = -1;
                }
            }   //CLOSE FILE
                close(fd);
        }
    }

    return retval;
```

STEPS:

- Split request and find number of words in request.
- If the format is not proper then free allocated memory.
- Check file exist in server
- If the file exist open it in read-only mode.
- Send client to file exist status.
- Lock the file to ensure synchronization.

- Read file into the Client_Response_Fifo.
- Unlock the file.

Ps : There is no function for read desired line. I could not manage it.

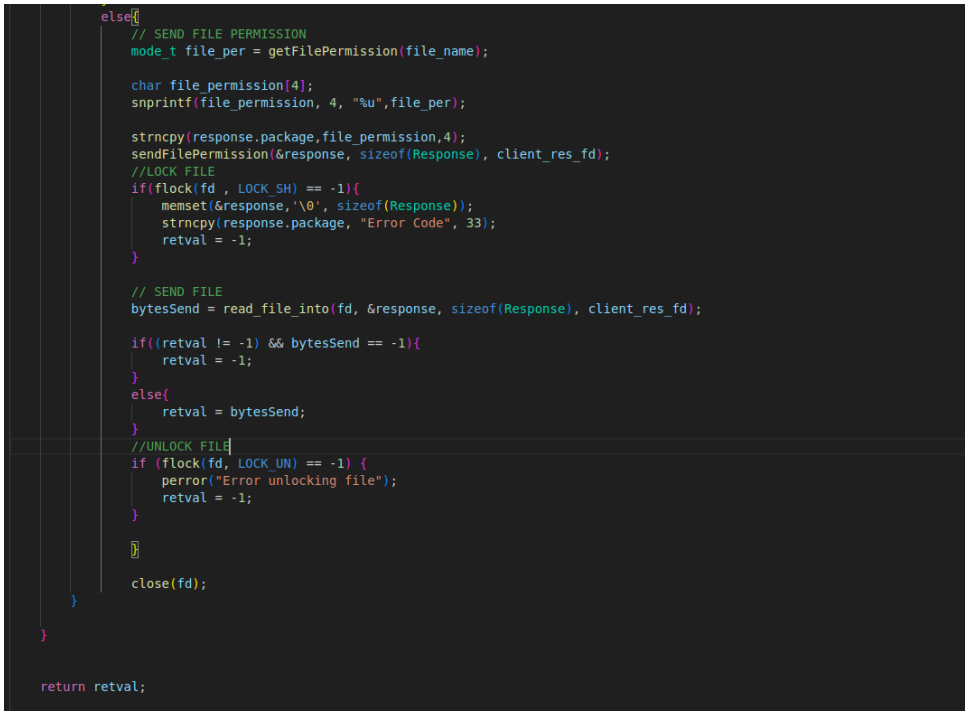
- **Handling ReadF in Client Side**

```
else if(strncmp(request.msg, "readF",5) == 0){


    /* finding downloadable files*/
    int word_count = get_word_count(request.msg);

    if(word_count == 2){

        if(read_file(client_res_fd, &response, sizeof(Response)) == -1){
            fprintf(stdout, "%s",response.package);
        }
        else if(strcmp(response.package, "Error Code") == 0){
            fprintf(stdout,"%s","There is no such a file in server\n");
        }
        else{
            if(fd != -1){
                if(read_file_into(client_res_fd, &response, sizeof(Response), STDOUT_FILENO) == -1){
                    perror("Error : ");
                }
            }
            else{
                perror("Error : ");
            }
        }

    }
```

- Check the format of the request is proper for readF or not
- Read file existing status.
- Read Client_Res_Fifo into the STDOUT.

- **Handling Download in Server Side**

```
int handle_Dowload(Request request, Response response, const int client_res_fd ){
    char* words[2];
    int i;
    int word_count;
    int retval = 0;
    int fd;
    char file_name[256];
    int bytesSend = 0;
    for(i = 0 ; i < 2; ++i ){
        words[i] = (char *) malloc(sizeof(char)* 256);
    }
    // SPLIT REQUEST
    word_count = split_string_into(request.msg, words,2);
    if( word_count == -1){
        retval = -1;
        for(i = 0 ; i < 2; ++i ){
            free(words[i]);
        }
    }
    else{
        // CHECK EXISTING OF FILE
        snprintf(file_name,256,"server_files/%s",words[1]);
        if(access(file_name, F_OK) == -1){
            memset(&response,'\0', sizeof(Response));
            strncpy(response.package, "Error Code", 33);
            write_to_file(client_res_fd, sizeof(Response), &response);
            retval = -1;
        }
        else{
            //OPEN FILE
            fd = open(file_name, O_RDONLY);
            if(fd == -1){
                retval = -1;
            }
```



```
        else{
            // SEND FILE PERMISSION
            mode_t file_per = getFilePermission(file_name);

            char file_permission[4];
            snprintf(file_permission, 4, "%u",file_per);

            strncpy(response.package,file_permission,4);
            sendFilePermission(&response, sizeof(Response), client_res_fd);
            //LOCK FILE
            if(flock(fd , LOCK_SH) == -1){
                memset(&response,'\0', sizeof(Response));
                strncpy(response.package, "Error Code", 33);
                retval = -1;
            }

            // SEND FILE
            bytesSend = read_file_into(fd, &response, sizeof(Response), client_res_fd);

            if((retval != -1) && bytesSend == -1){
                retval = -1;
            }
            else{
                retval = bytesSend;
            }
            //UNLOCK FILE
            if (flock(fd, LOCK_UN) == -1) {
                perror("Error unlocking file");
                retval = -1;
            }

            }

            close(fd);
    }

}


return retval;
```

STEPS:

- Split Request into proper format
- Check file exists in server
- Open file and get file permissions of desired format
- Send file permissions
- Lock file to ensure synchraniaton
- Send file to the Client_Response_Fifo
- Unlock file

- **Handling Download in Client Side**

```
else if(strncmp(request.msg, "download", 8) == 0){

    /* finding downloadable files*/
    char *words[2];
    int i;
    for (i = 0; i < 2; i++)
    {
        words[i] = (char*)malloc((sizeof(char)*256));
    }
    // SPLIT REQUEST
    int word_count = split_string_into(request.msg, words,2);

    if(word_count == -1){
        for (i = 0; i < 2; i++)
        {
            free(words[i]);
        }
    }
    // Check file format

    if(word_count != -1){

        char  file_name[256];
        snprintf(file_name, 256, "download/%s",words[1]);

        // READ CLIENT RESPONSE FIFO TO FIND FILE EXIST OR NOT
        if(read_file(client_res_fd, &response, sizeof(Response)) == -1){
            fprintf(stdout, "%s",response.package);
        }
        else if(strcmp(response.package, "Error Code") == 0){
            fprintf(stdout,"%s","There is no such a file in server\n");
        }
```

```
    // GET FILE PERMISSON FROM CLIENT_RESOONSE_FIFO
    else{

        mode_t file_per = (unsigned int)atoi(response.package);
        // CREATE FILE IF NOT EXIST AND READ FIFO INTO THIS FILE
        fd = open(file_name, O_CREAT|O_WRONLY,file_per);
        if(fd != -1){
            if(read_file_into(client_res_fd, &response, sizeof(Response), fd) == -1){
                perror("Error : ");
            }

            close(fd);
        }
        else{
            perror("Error : ");
        }
    }

}
```

STEPS:

- Split request message to desired format.
- Read file existing status
- Create File into download directory if does not exist
- Read Client_Response_Fifo into the file

- **Handling Upload in Server Side**

```c
int handle_Upload(Request request, Response response, const int client_res_fd_wr){
    /* finding downloadable files*/

        char *words[2];
        int i;
        int fd;
        int retval = 0;
        for (i = 0; i < 2; i++)
        {
            words[i] = (char*)malloc((sizeof(char)*256));
        }
        // get word count and split request into desired format
        int word_count = split_string_into(request.msg, words,2);
        // Check file format
        //first we need to find file permission

        if(word_count != -1){

            char  file_name[256];
            snprintf(file_name, 256, "server_files/%s",words[1]);
            // read CLEINT RESPONSE_FIFO_WR TO CHECK FILE EXIST OR NOT
            if(read_file(client_res_fd_wr, &response, sizeof(Response))== -1){
                retval = -1;
                perror("Error : ");
            }
            else if(strncmp(response.package, "Error Code", sizeof(Response)) == 0  ){
                retval = -1;
            }
        }
        else{
            // GET FILE PERMISSON AND CREATE OR OPEN FILE IN SERVER
            mode_t file_per = (unsigned int)atoi(response.package);

            fd = open(file_name, O_CREAT|O_WRONLY,file_per);

            if(fd != -1){
                // LOCK THE FILE
                 if(flock(fd , LOCK_SH) == -1){
                   memset(&response,'\0', sizeof(Response));
                   strncpy(response.package, "Error Code", 33);
                   retval = -1;
                   }
                // READ FIFO INTO THE UPLOADED FILE
                if(read_file_into(client_res_fd_wr, &response, sizeof(Response), fd) == -1){
                    perror("Error : ");
                    retval = -1;

                }
                // UNLOCK FILE
                if (flock(fd, LOCK_UN) == -1) {
                  perror("Error unlocking file");
                    retval = -1;
                }

                close(fd);
            }
            else{
                perror("Error : ");
                retval = -1;

            }
        }
    }
    return retval;
```

STEPS:

- Split request message to obtain desired format
- Read Client REsponse_FIFO_Wr to check file existence
- Get file permission
- Lock the file
- Write uploaded file to newly created file
- Unlock the file

- **Handling Upload in Client Side**

```c
int handleUpload(Request request, Response response, const int client_res_fd_wr){
    char* words[2];
    int i;
    int word_count;
    int retval = 0;
    int fd;
    for(i = 0 ; i < 2; ++i ){
        words[i] = (char *) malloc(sizeof(char)* 256);
    }
    // split request into desired format
    word_count = split_string_into(request.msg, words,3);
    if( word_count == -1){
        retval = -1;
        for(i = 0 ; i < 2; ++i ){
            free(words[i]);
        }
    }
    else{
        // check file is exist in client or not
        if(access(words[1], F_OK) == -1){
            memset(&response,'\0', sizeof(Response));
            strncpy(response.package, "Error Code", 33);
            write_to_file(client_res_fd_wr, sizeof(Response), &response);
            retval = -1;
            perror("Error : ");
        }
        else{
            // open file as read
            fd = open(words[1], O_RDONLY);
            if(fd == -1){
                retval = -1;
            }
            else{
                // get file permission and send to the server
                mode_t file_per = getFilePermission(words[1]);

                char file_permission[4];
                snprintf(file_permission, 4, "%u",file_per);

                strncpy(response.package,file_permission,4);
                sendFilePermission(&response, sizeof(Response), client_res_fd_wr);
                // read file into the CLIENT_RESPONSE_FIFO_WR For server
                if(read_file_into(fd, &response, sizeof(Response), client_res_fd_wr) == -1){
                    retval = -1;
                }
                }
                close(fd);
```

STEPS:

- Split the request message
- Check the file existence
- Open the file as readed
- Get the file permissons and send to server
- Read file into CLIENT_RESPONSE_FIFO_Wr to be readed in server
-
- **Handling Quit**

```c
    }
    else if((strcmp(request.msg, "quit")) == 0){
        //cleanUp(client_res_fd, client_res_fd_wr);
        sem_wait(client_num_sem);
        --(*client_num_shm);
        sem_post(client_num_sem);
        close(client_res_fd);
        close(client_res_fd_wr);
        close(client_req_fd);
        fprintf(stderr,">> Client with %d pid disconnected\n",request.pid);
        exit(EXIT_SUCCESS);
    }
```

STEPS:

- There is a mutex to avoid race conditions to decrement client number.
- Close fifos, instead of SERVER_FIFO
- Then exit. SIG_CHLD signal is handled here. If the child exist , in SIG_CHLD handler, Parent wait it.

TEST