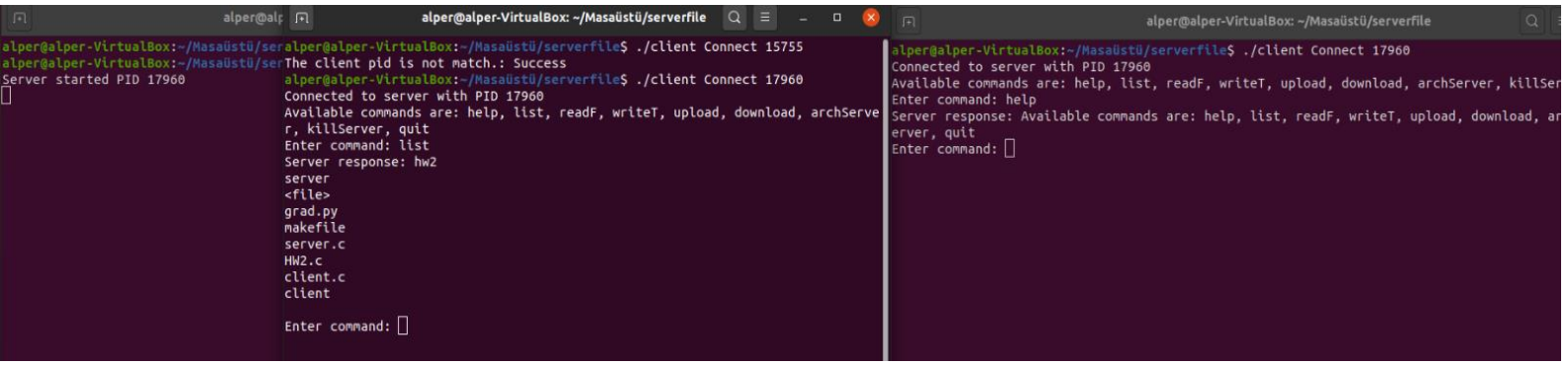
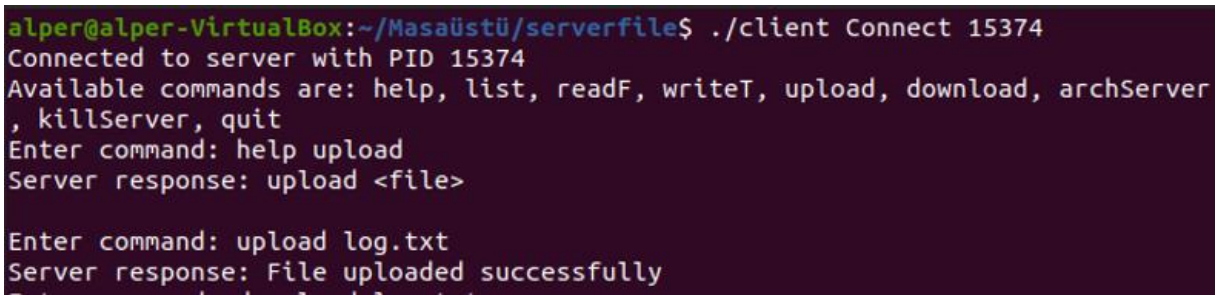
Gebze Technical University
Department of Computer Engineering
CSE344 - Spring 2024
Midterm Project
AHMET ALPER UZUNTEPE
1901042669



The system comprises a server program and a client program, which communicate via shared memory(Queue with Semaphore). The server manages client requests, processes them, and sends back responses. The client sends commands to the server and receives corresponding responses.We learned before midterm ,share memory so ı thought we can use share memory. Before last week (midterm) we learned semaphore also. So that ı am going to this Project based on Semaphore(Synchronization-Prevent race condution) and share Memory .



THE LIBRARIES

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <errno.h>
#include <semaphore.h>
#include <sys/sem.h>
```

- input/output (stdio.h),
- standard library functions (stdlib.h),
- system calls (unistd.h),
- string manipulation (string.h),
- shared memory functions (sys/shm.h),
- waiting for processes (sys/wait.h),
- file operations (sys/stat.h, fcntl.h),
- directory operations (dirent.h),
- and error handling (errno.h),
- semaphore-related header files (sys/sem.h)

STRUCTURE OF SHARED MEMORY

```c
// structure for shared data
typedef struct {
    pid_t client_pid;   // store the PID
    char command[CMD_SIZE]; // store the
    int counter_client; // counter for ho
    int max_clients; //max client number
} SharedData;
```

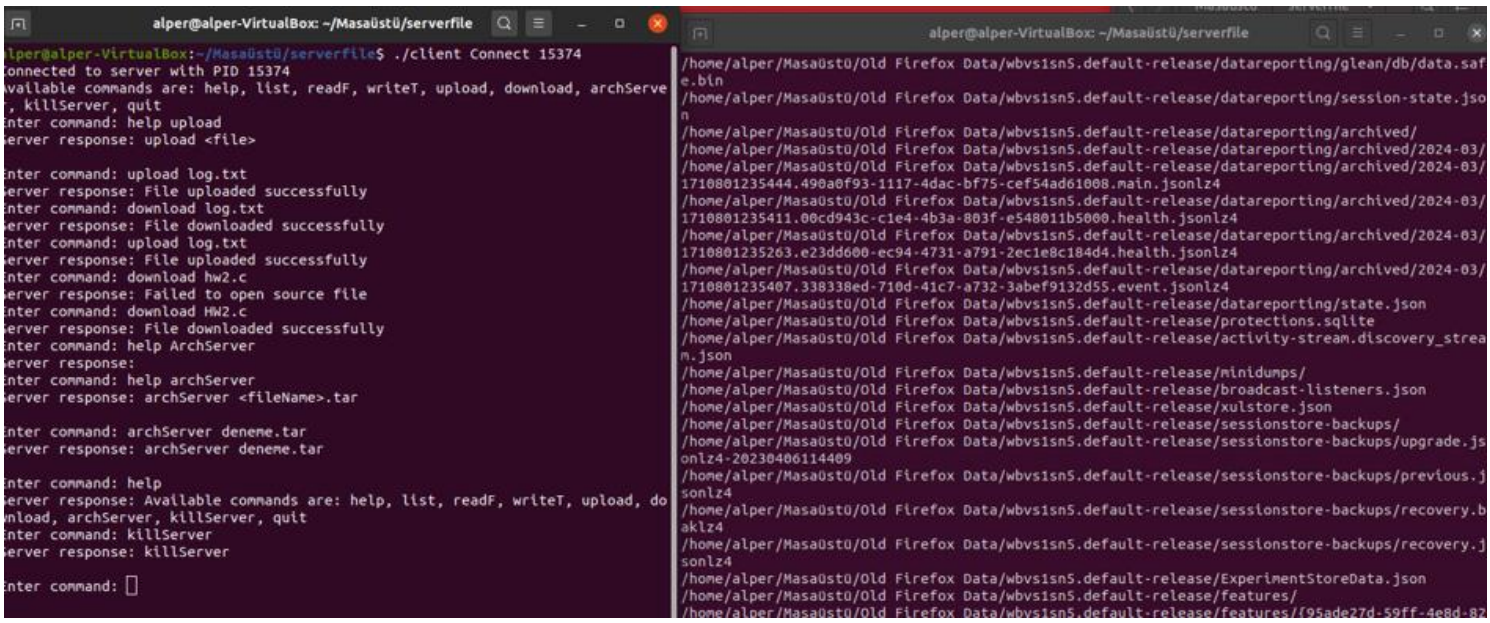
- client_pid ,store the PID of the client
- char command,store the command sent by the client actually it is the communication between client and server with like response
- counter_client,counter for how many clients is working it increase and decrease
- max_clients ,max client number the given as argument

SERVER SIDE

1)SEMAPHORE

```c
int initialize_semaphore(int *sem_id) {
    *sem_id = semget(ftok("server", 'S'), 1, IPC_CREAT | IPC_EXCL | 0666);
    if (*sem_id == -1) {
        return -1;
    }

    if (semctl(*sem_id, 0, SETVAL, 1) == -1) { // set the semaphore value
        return -1;
    }

    return 0;
}

void destroy_semaphore(int sem_id) {
    semctl(sem_id, 0, IPC_RMID, NULL);
}
```

Initialize_semaphore creates a semaphore with an initial value of 1 and returns its identifier, while destroy_semaphore removes a semaphore from the system. These functions are essential for managing semaphores.



2)HANDLE CLIENT FUNCTION

```c
void handle_client(SharedData *shared_memory, char *dirname,int sem_id)
```

- It takes three arguments: a pointer to SharedData structure (shared_memory), a character array (dirname), and an integer (sem_id).

```
//process client request (access shared memory)

char commands[CMD_SIZE];
strcpy(commands, shared_memory->command); // copy the command from shared memory
```

- Declares a character array commands of size CMD_SIZE to store the command received from the shared memory.
- Copies the command from the shared_memory structure into the commands array using strcpy.

```
char response[CMD_SIZE];
response[0] = '\0' ; // initialize the response variable
```

- Declares a character array response of size CMD_SIZE to store the response to the client.
- Initializes the response as an empty string.(I did it at the beginning because the this handle_client function in a loop in main function.It must be empty.)

help

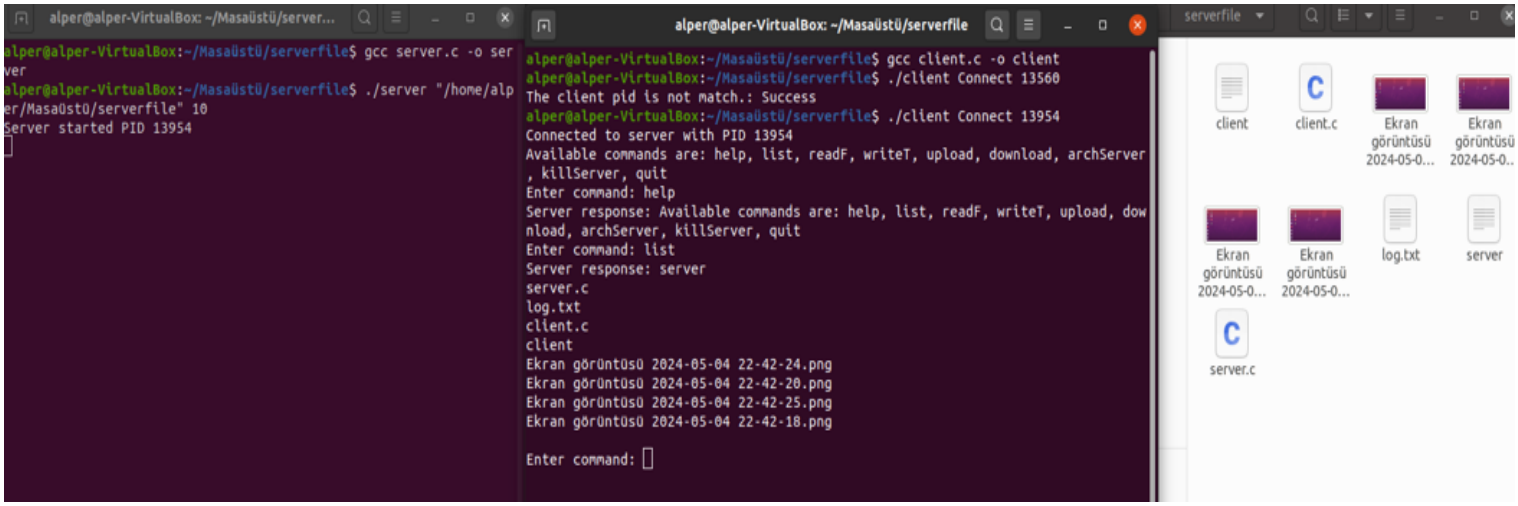
```
if (strncmp(commands, "help\n", sizeof(commands)) == 0)
{
    //printf("Received command: help\n");
    strcpy(response, "Available commands are: help, list, readF, writeT, upload, download, archServer, killServer, quit");
}
```

- Checks if the received command is "help\n".
- If it is, it sets the response to a string listing available commands.

list

```c
else if (strcmp(shared_memory->command, "list\n") == 0)
{
    DIR *dir;
    struct dirent *entry;
    if ((dir = opendir(dirname)) != NULL)// open the directory for reading
    {
        while ((entry = readdir(dir)) != NULL)// iterate each entry in the directory
        {
            if (entry->d_type == DT_REG)// check if the entry is a regular file
            {
                strcat(response, entry->d_name);// append the filename to the response string and after that we need newline
                strcat(response, "\n");
            }
        }
        closedir(dir);// close the directory
    }
    else
    {
        perror("Failed to open directory");// print error message if failed to open dir
    }
}
```

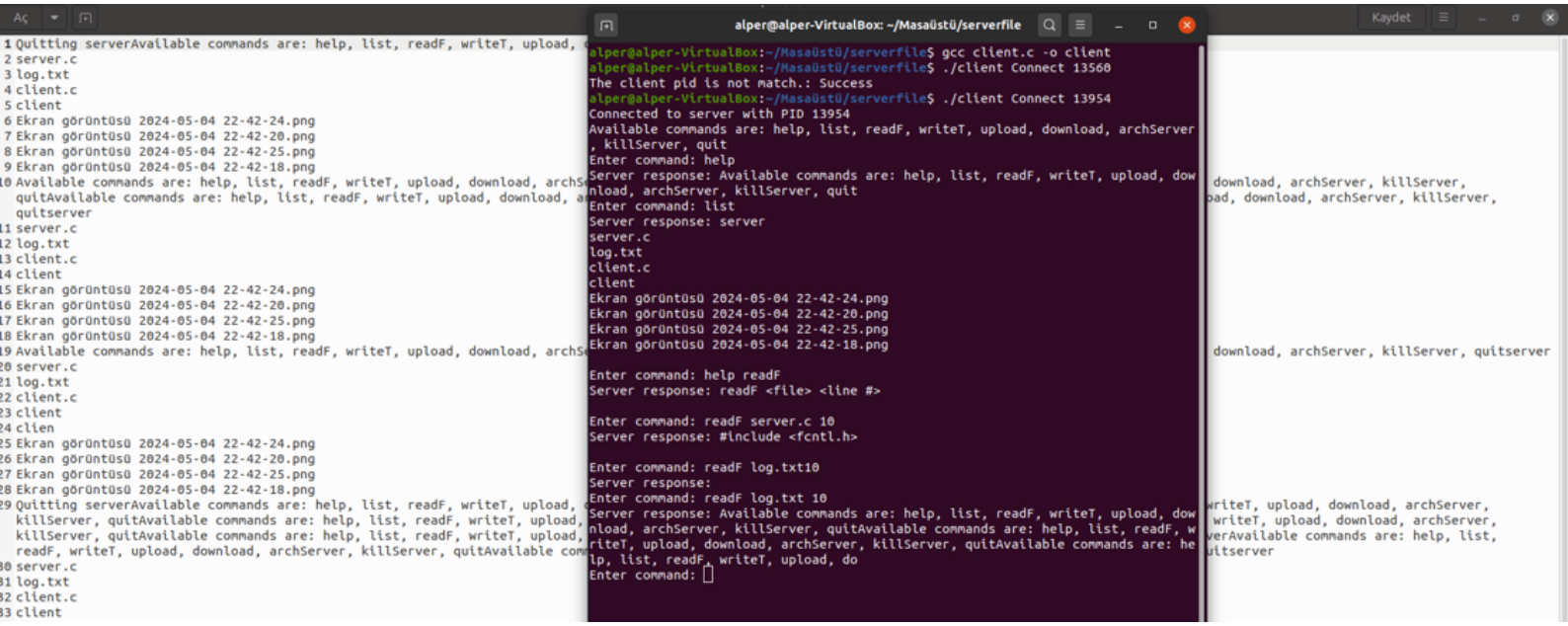
- Checks if the received command is "list\n".
- If it is, it opens the directory specified by dirname.
- Iterates through each entry in the directory.
- If the entry is a regular file (DT_REG), it appends the filename to the response string followed by a newline character.
- Finally, it closes the directory.



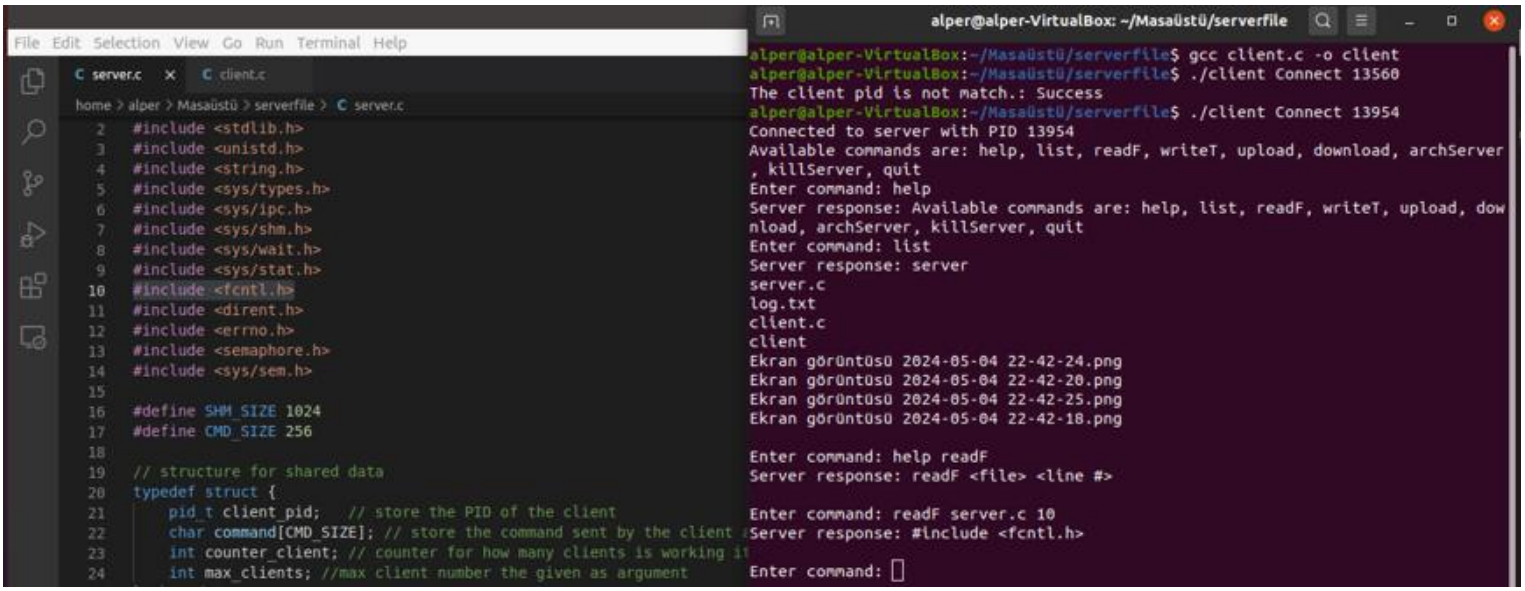
readF

```c
else if (strncmp(shared_memory->command, "readF\n", 5) == 0)
{
    char filename[CMD_SIZE]; // filename and line number from the command
    int line_number = 0;
    sscanf(shared_memory->command, "readF %s %d", filename, &line_number);
    FILE *file = fopen(filename, "r");// open the file for reading
    if (file)
    {
        if (line_number > 0)
        {
            char line[CMD_SIZE];// read each line until reaching line number
            for (int i = 0; i < line_number; i++)
            {
                if (fgets(line, sizeof(line), file) == NULL)
                {
                    strcpy(response, "Line number exceeds file length");
                    break;
                }
            }
            strcpy(response, line);// copy the last read line to the response
        }
        else// read the content of the file if line number is non-positive
        {

            fseek(file, 0, SEEK_END);// move pointer to the end
            long file_size = ftell(file);// get the file size
            fseek(file, 0, SEEK_SET);// reset  pointer to the began
            if (file_size >= CMD_SIZE)
            { // check for potential truncation
                perror("File content truncated");
                file_size = CMD_SIZE - 1;
            }
            fread(response, 1, file_size, file);
            response[file_size] = '\0';// null at the end
        }
        fclose(file);//close file
    }
    else
    {
        perror("Failed to open file");
    }
}
```

- Checks if the received command starts with "readF\n" (reads a file).
- If the condition is true, it means the client requested to read from a file.
- Uses sscanf to parse the command stored in shared_memory->command. It extracts the filename and line number from the command string and stores them in filename and line_number, respectively.



- If the line number exceeds the length of the file, it sets the response to indicate that the line number exceeds the file length.
- Otherwise, it copies the last read line into the response.
- It reads the file content using fread and stores it in response.
- I use fgets() function in a loop for reading file line by line.



writeT

```c
else if (strncmp(shared_memory->command, "writeT\n", 6) == 0)
{
    char filename[CMD_SIZE];// filename, line number, and string from the command
    int line_number;
    char string[CMD_SIZE];
    sscanf(shared_memory->command, "writeT %s %d %[^\n]", filename, &line_number, string);
    FILE *file = fopen(filename, "a");// open the file for appending
    if (file)
    {
        if (line_number > 0)
        {
            char temp[CMD_SIZE];
            int current_line = 0;
            while (fgets(temp, sizeof(temp), file))// iteration
            {
                current_line++;
                if (current_line == line_number)// if the current line matches the given line number
                {
                    fprintf(file, "%s\n", string);
                    strcpy(response, "String written to file");
                    break;
                }
            }
            if (current_line != line_number)// if the given line number exceeds the file length
            {
                strcpy(response, "Line number exceeds file length");
            }
        }
        else // write the string to the file without considering line numbers
        {
            fprintf(file, "%s\n", string);
            strcpy(response, "String written to file");
        }
        fclose(file);
    }
    else
    {
        perror("Failed to open file");
    }
}
```

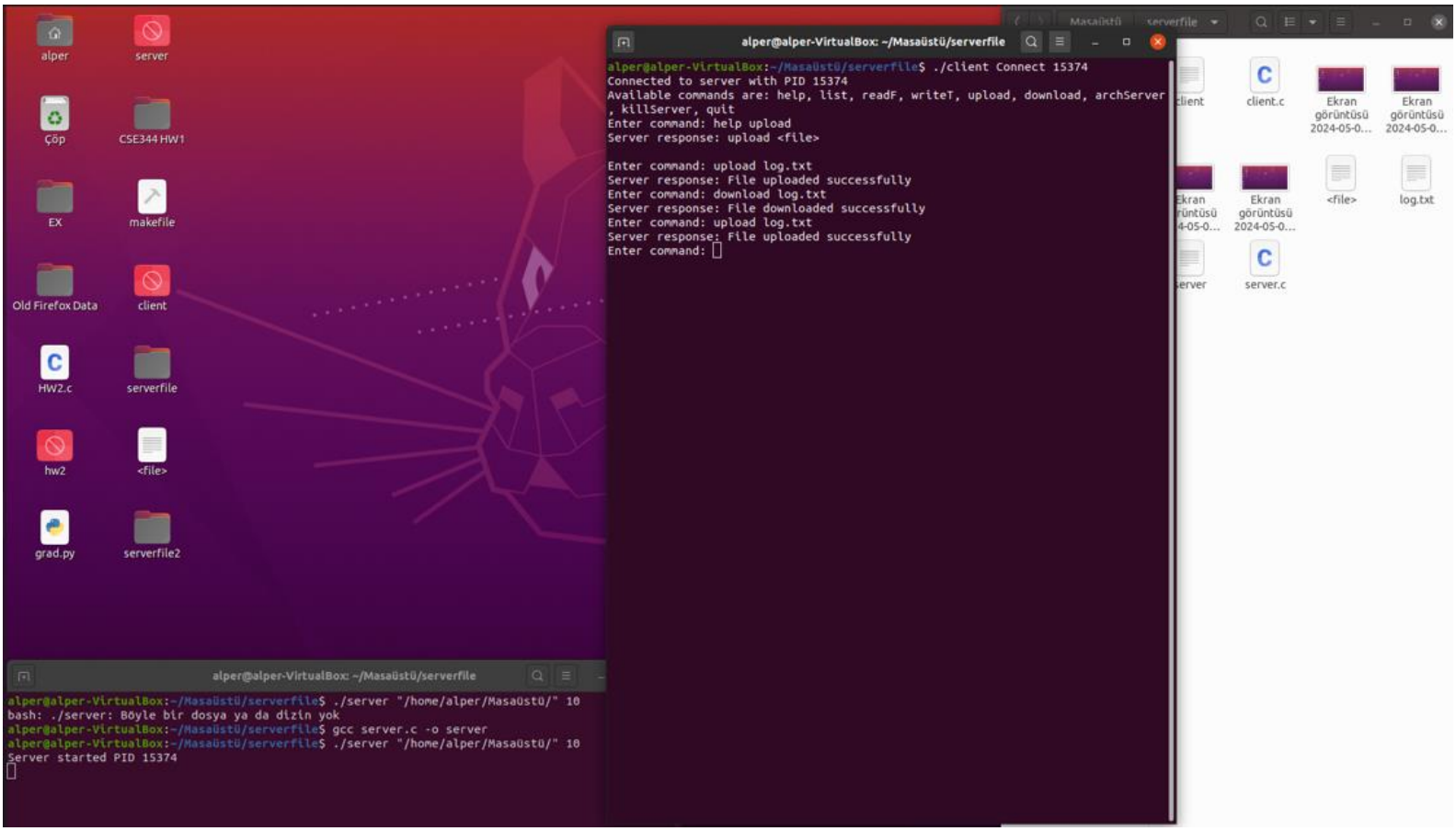
- Checks if the received command starts with "writeT\n" (writes to a file).
- If the condition is true, it means the client requested to write to a file.
- Uses sscanf to parse the command stored in shared_memory->command. It extracts the filename, line number, and string from the command string and stores them in filename, line_number, and string, respectively.Opens the file specified by filename for appending.
- I use fgets() function in a loop for reading file line by line.
- If the line number exceeds the length of the file, it sets the response to indicate that the line number exceeds the file length.
- Otherwise, it writes the string to the specified line in the file using fprintf and sets the response to indicate that the string was written successfully.

upload

```c
else if (strncmp(shared_memory->command, "upload\n", 6) == 0)
{
    char filename[CMD_SIZE];// filename from the command
    sscanf(shared_memory->command, "upload %s", filename);

    char source[CMD_SIZE];// source path (current directory)
    int source_len = snprintf(source, CMD_SIZE, "%s/%s", ".", filename); // Use snprintf
    if (source_len >= CMD_SIZE)
    {
        perror("Source path truncated");
        return;
    }

    char dest[CMD_SIZE];// destination path (specified directory)
    int dest_len = snprintf(dest, CMD_SIZE, "%s/%s", dirname, filename); // Use snprintf
    if (dest_len >= CMD_SIZE)
    {
        perror("Destination path truncated");
        return;
    }

    int source_fd = open(source, O_RDONLY);// open the source file for reading
    if (source_fd == -1)
    {
        perror("Failed to open source file");
        strcpy(response, "Failed to open source file");
    }
    else
    {
        int dest_fd = open(dest, O_WRONLY | O_CREAT | O_TRUNC, 0644);// destination file
        if (dest_fd == -1)
        {
            perror("Failed to create destination file");
            strcpy(response, "Failed to create destination file");
        }
        else
        {
            char buffer[4096];
            ssize_t bytes_read;
            while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0)// read fr
            {
                ssize_t bytes_written = write(dest_fd, buffer, bytes_read);
                if (bytes_written != bytes_read)
                {
                    perror("Failed to write to destination file");
                    strcpy(response, "Failed to write to destination file");
                    break;
                }
            }
            if (bytes_read == -1)
            {
                perror("Failed to read from source file");
                strcpy(response, "Failed to read from source file");
            }
            else
            {
                strcpy(response, "File uploaded successfully");
            }
            close(dest_fd);// close the derstination file
        }
        close(source_fd); // close the source file
    }
}
```

- Checks if the received command starts with "upload\n".
- Declares a character array filename to store the filename extracted from the command.

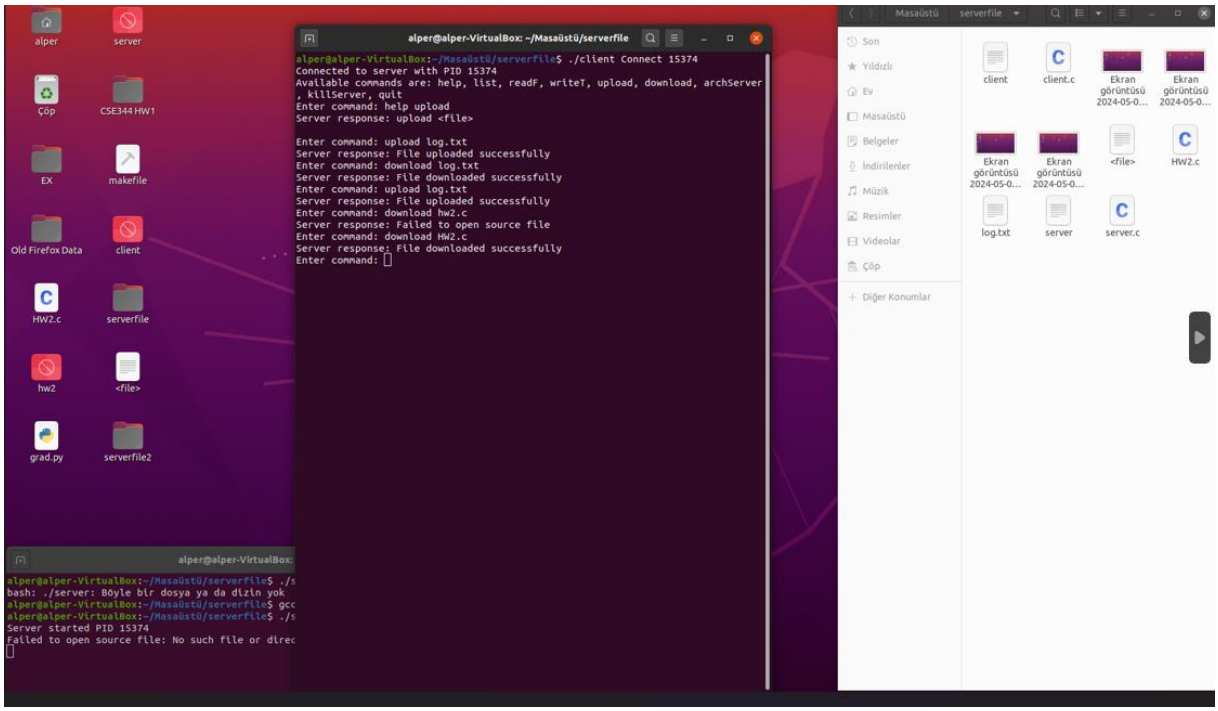
- Uses sscanf to parse the command stored in shared_memory->command. It extracts the filename after the "upload" keyword and stores it in the filename variable.
- Constructs the source path by concatenating the current directory (".") with the filename obtained from the command.
- Uses snprintf to ensure that the source path doesn't exceed the size of source.



- If the source path exceeds the size, it prints an error message indicating that the source path was truncated and returns from the function.
- Constructs the destination path by concatenating the specified directory (dirname) with the filename obtained from the command.
- Uses snprintf to ensure that the destination path doesn't exceed the size of dest.
- If the destination path exceeds the size, it prints an error message indicating that the destination path was truncated and returns from the function.
- Opens the source file specified by the source path for reading (O_RDONLY).

- If the source file couldn't be opened, it prints an error message and sets the response to indicate the failure.
- Opens the destination file specified by the destination path for writing (O_WRONLY | O_CREAT | O_TRUNC). If the file doesn't exist, it creates it. If it exists, it truncates it to zero length.
- If the destination file couldn't be created, it prints an error message and sets the response to indicate the failure.
- Reads data from the source file in chunks (4096 bytes at a time) and writes it to the destination file.
- If there is an error while writing to the destination file, it prints an error message, sets the response to indicate the failure, and breaks out of the loop.
- If there is an error while reading from the source file, it prints an error message and sets the response to indicate the failure.
- If the file is uploaded successfully, it sets the response to indicate success.

download



- Checks if the received command starts with "download\n".
- Declares a character array filename to store the filename extracted from the command.
- Uses sscanf to parse the command stored in shared_memory->command. It extracts the filename after the "download" keyword and stores it in the filename variable.

```
else if (strncmp(shared_memory->command, "download\n", 8) == 0)
{
    char filename[CMD_SIZE];// filename from the command
    sscanf(shared_memory->command, "download %s", filename);
    char source[CMD_SIZE];
    int source_len = snprintf(source, CMD_SIZE, "%s/%s", dirname, filename); // use

    if (source_len >= CMD_SIZE)// source path
    {
        perror("Source path truncated");
        return;
    }

    char dest[CMD_SIZE];// destination path
    int dest_len = snprintf(dest, CMD_SIZE, "%s/%s", ".", filename); // use snprint
    if (dest_len >= CMD_SIZE)
    {
        perror("Destination path truncated");
        return;
    }

    int source_fd = open(source, O_RDONLY);// open the source file for reading
    if (source_fd == -1)
    {
        perror("Failed to open source file");
        strcpy(response, "Failed to open source file");
    }
    else
    {
        int dest_fd = open(dest, O_WRONLY | O_CREAT | O_TRUNC, 0644);// destination
        if (dest_fd == -1)
        {
            perror("Failed to create destination file");
            strcpy(response, "Failed to create destination file");
        }
        else
        {
            char buffer[4096];
            ssize_t bytes_read;
            while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0)// re
            {
                ssize_t bytes_written = write(dest_fd, buffer, bytes_read);
                if (bytes_written != bytes_read)
                {
                    perror("Failed to write to destination file");
                    strcpy(response, "Failed to write to destination file");
                    break;
                }
            }
            if (bytes_read == -1)
            {
                perror("Failed to read from source file");
                strcpy(response, "Failed to read from source");
            }
            else
            {
                strcpy(response, "File downloaded successfully");
            }
            close(dest_fd);// close the destination file
        }
        close(source_fd);// close the spurce file
    }
}
```

- Constructs the source path by concatenating the specified directory (dirname) with the filename obtained from the command using snprintf.
- source_len holds the length of the constructed source path.
- Checks if the length of the constructed source path exceeds the size limit.
- If true, it prints an error message indicating that the source path was truncated and returns from the function.
- Constructs the destination path by concatenating the current directory (".") with the filename obtained from the command using snprintf.
- dest_len holds the length of the constructed destination path.
- Checks if the length of the constructed destination path exceeds the size limit.
- If true, it prints an error message indicating that the destination path was truncated and returns from the function.
- Opens the source file specified by the source path for reading (O_RDONLY).
- If the source file couldn't be opened, it prints an error message and sets the response to indicate the failure.
- Opens the destination file specified by the destination path for writing (O_WRONLY | O_CREAT | O_TRUNC). If the file doesn't exist, it creates it. If it exists, it truncates it to zero length.
- If the destination file couldn't be created, it prints an error message and sets the response to indicate the failure.
- Reads data from the source file in chunks (4096 bytes at a time) and writes it to the destination file.
- If there is an error while writing to the destination file, it prints an error message, sets the response to indicate the failure, and breaks out of the loop.
- If there is an error while reading from the source file, it prints an error message and sets the response to indicate the failure.

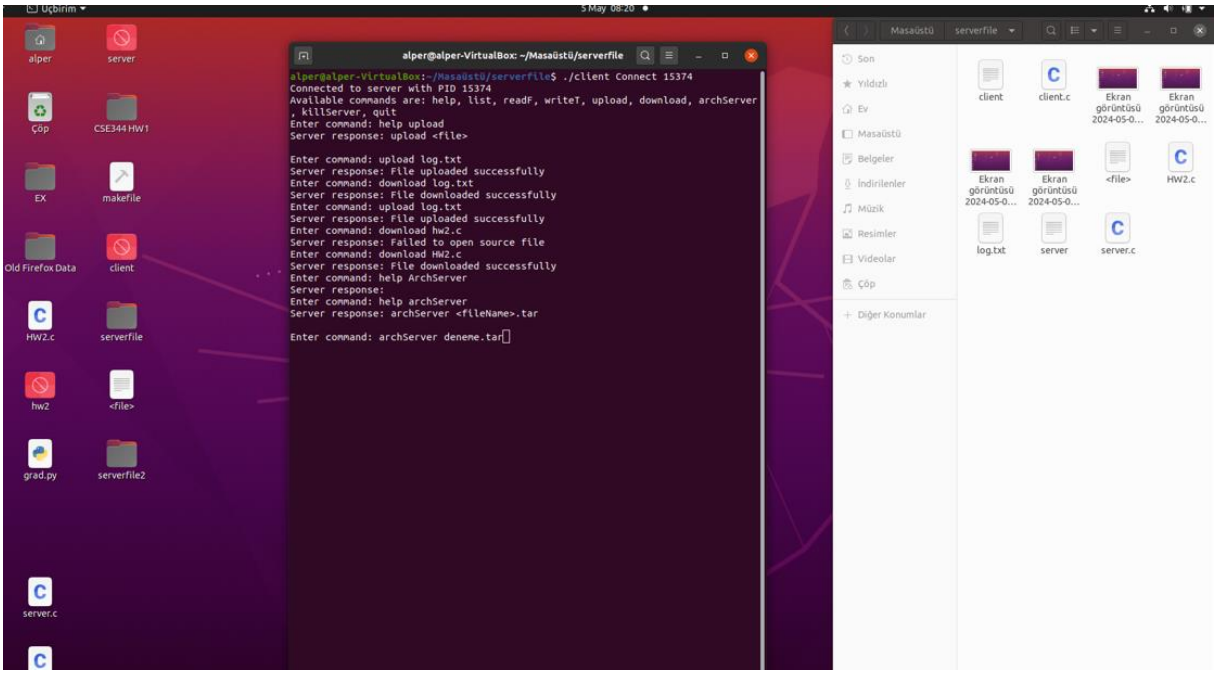
- If the file is downloaded successfully, it sets the response to indicate success.Closes the destination file and the source file after the download process is completed.

archServer

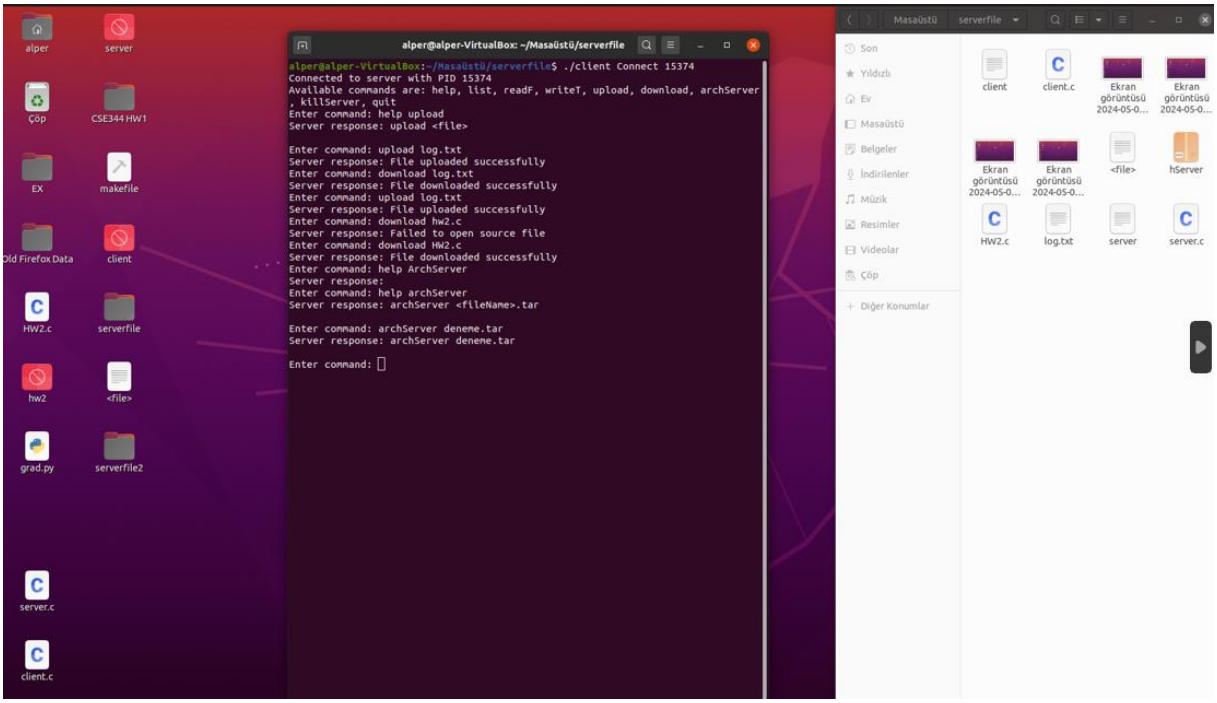
```c
else if (strncmp(shared_memory->command, "archServer", 10) == 0)
{
    char filename[CMD_SIZE];
    int space_after_cmd = strcspn(filename, "\n"); // space after command
    //printf("deneme\n");

    sscanf(shared_memory->command + space_after_cmd + 1, "%s", filename);
    //printf("deneme 3 : %s \n",filename);;
    pid_t child_pid = fork();
    if (child_pid == -1)
    {
        perror("Failed to fork process");
        strcpy(response, "Failed to create archive");

        return;
    }

    else if (child_pid == 0)
    {                           // child process
        char tar_cmd[CMD_SIZE];
        int written_bytes = snprintf(tar_cmd, CMD_SIZE, "tar -cvf %s %s/*", filename, dirname);
        if (written_bytes >= CMD_SIZE)
        {
            perror("tar command string truncated");
            exit(EXIT_FAILURE);
        }

        // execvp to avoid potential shell injection vulnerabilities
        execvp("tar", (char *[]){"tar", "-cvf", filename, dirname, NULL});
        perror("Failed to execute tar command");
        exit(EXIT_FAILURE); // if execvp fails, exit child process
    }

     // Parent process
    //  printf("succes fork \n");
        int wait_status;
        waitpid(child_pid, &wait_status, 0);
        if (WIFEXITED(wait_status) && WEXITSTATUS(wait_status) == 0)
        {
            strcpy(response, "Server files archived successfully");
        }
        else
        {
            strcpy(response, "Failed to create archive");
        }

}
```

- Checks if the received command starts with "archServer".
- Declares a character array filename to store the filename extracted from the command.
- strcspn calculates the length of the initial segment of filename that consists of characters not in the string "\n". However, at this point, filename is uninitialized, which leads to undefined behavior.

- Uses sscanf to extract the filename from the command. It starts parsing from shared_memory->command after the newline character (\n) and stores the result in filename.



(BEFORE)



(AFTER)

- Forks a child process. If the fork fails (fork() returns -1), it prints an error message, sets the response to indicate failure, and returns from the function.
- Inside the child process:
- Declares a character array tar_cmd to store the command to execute the tar utility.

- Uses snprintf to construct the command string. It writes formatted output to the string tar_cmd, ensuring that it doesn't exceed the size of CMD_SIZE.
- Executes the tar command using execvp. This replaces the current process with a new process as specified by the command.
- If execvp fails (returns -1), it prints an error message, indicating the failure, and exits the child process with failure status.
- In the parent process:
- Uses waitpid to wait for the child process to complete and store its exit status in wait_status.
- Checks if the child process exited normally (WIFEXITED) and if its exit status indicates success (WEXITSTATUS is 0). If true, it sets the response to indicate success; otherwise, it sets it to indicate failure.

killServer(Works like Ctrl+C)

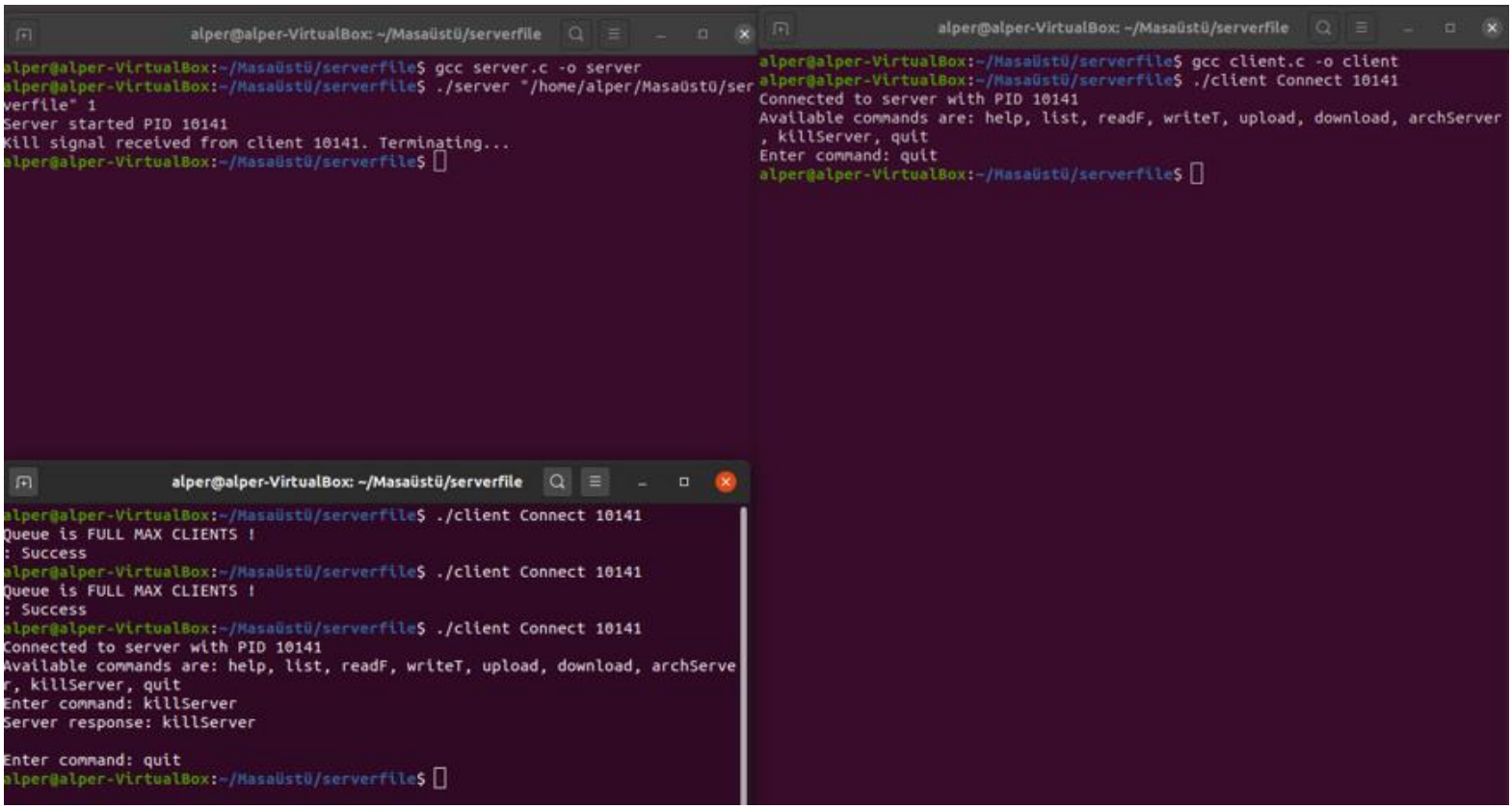
```
else if (strcmp(shared_memory->command, "killServer\n") == 0)
{
    printf("Kill signal received from client . Terminating...\n");
    strcpy(response, "Kill server");
    exit(EXIT_SUCCESS);//it is works like ctrl+c . like clean shut down .
}
```

- This line checks if the received command stored in shared_memory->command is equal to "killServer\n".
- If the condition is true, this line prints a message indicating that a kill signal has been received from the client, indicating the server to terminate.
- Copies the string "Kill server" to the response variable. This is likely to inform the client that the server is being terminated.
- This line exits the program with a successful status (EXIT_SUCCESS). It terminates the program immediately, as if a clean shutdown process is initiated, similar to pressing Ctrl+C. This effectively stops the server process.

quit

```c
else if (strcmp(shared_memory->command, "quit\n") == 0)//quit for clients
{
    strcpy(response, "Quitting server");
    shared_memory->counter_client = shared_memory->counter_client -1 ; //decrease counter
    //kill(-getpgrp(), SIGTERM);
}
```

- Checks if the received command is "quit\n".
- Copies the string "Quitting server" into the response buffer. This message indicates that the server is going to quit.
- Decrements the value of shared_memory->counter_client by 1. This counter likely tracks the number of connected clients or some other relevant metric.



Usage Directory

```
// help for operations return response for directory
else if (strcmp(shared_memory->command, "help readF\n") == 0)
{
    strcpy(response, "readF <file> <line #>\n");
}
else if (strcmp(shared_memory->command, "help writeT\n") == 0)
{
    strcpy(response, "writeT <file> <line #> <string>\n");
}
else if (strcmp(shared_memory->command, "help upload\n") == 0)
{
    strcpy(response, "upload <file>\n");
}
else if (strcmp(shared_memory->command, "help download\n") == 0)
{
    strcpy(response, "download <file>\n");
}
else if (strcmp(shared_memory->command, "help archServer\n") == 0)
{
    strcpy(response, "archServer <fileName>.tar\n");
}
```

- This is for direction for usage .
- Usages stores in response and client takes it .

```
// Write the response to the log file
int log_fd = open("log.txt", O_WRONLY | O_CREAT | O_APPEND, 0644);
if (log_fd == -1) {
    perror("Failed to open log file");
    exit(EXIT_FAILURE);
}

if (write(log_fd, response, strlen(response)) == -1) {
    perror("Failed to write to log file");
}
close(log_fd);
```

- Opens a file named "log.txt" in write-only mode (O_WRONLY).
- Checks if the file opening was successful by comparing log_fd with -1.
- Writes the content of the response buffer to the file represented by log_fd.

```
// copy the response back to the shared memory command field
strcpy(shared_memory->command, response);
```

- Copies the content of the response buffer to the command field of the shared memory shared_memory.

Main(server.c)

```c
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <dirname> <max. #ofClients>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *dirname = argv[1];

    // create directory (if it doesn't exist)
    mkdir(dirname, 0777);

    // key for semaphore and shared memory
    key_t key = ftok("server", 'R');

    // create semaphore
    int sem_id;
    if (initialize_semaphore(&sem_id) == -1) {
        perror("Failed to initialize semaphore");
        exit(EXIT_FAILURE);
    }

    // create shared memory segment
    int shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("Failed to create shared memory segment");
        exit(EXIT_FAILURE);
    }

    // attach shared memory
    SharedData *shared_memory = (SharedData *)shmat(shm_id, NULL, 0);
    if (shared_memory == (void *)-1) {
        perror("Failed to attach shared memory");
        exit(EXIT_FAILURE);
    }

    printf("Server started PID %d\n", getpid());
    shared_memory->client_pid = getpid();
    shared_memory->counter_client = 0;
    shared_memory->max_clients = atoi(argv[2]);

    // main server loop
    while (1) {
        handle_client(shared_memory, dirname,sem_id);
        sleep(1); // sleep for server response
    }


    kill(-getpgrp(), SIGTERM);// cleanup
    shmdt(shared_memory);
    shmctl(shm_id, IPC_RMID, NULL);
    destroy_semaphore(sem_id);

    return 0;
}
```

- The main function, the entry point of the program, takes two arguments: argc, the number of command-line arguments, and argv, an array of strings containing the command-line arguments.
- Attempts to create a directory with the name specified by dirname if it doesn't already exist.
- The mode 0666 specifies full permissions for user, group, and others.
- Generates IPC key based on a file path (in this case, "server") and a project identifier ('R').
- This key will be used to identify the semaphore and shared memory segments.
- Initializes a semaphore by calling the initialize_semaphore function.
- If the initialization fails (returns -1), it prints an error message and exits the program with a failure status.
- Creates a shared memory segment using the IPC mechanism with the given key (key) and size (SHM_SIZE).
- Attaches the shared memory segment identified by shm_id to the address space of the calling process.
- Prints a message indicating that the server has started, along with its process ID.
- Initializes fields of the SharedData structure stored in the shared memory segment: client_pid with the process ID, counter_client to 0, and max_clients with the maximum number of clients specified as the third command-line argument converted to an integer.(For second argumant ,count the clients)
- Enters the main server loop, where it repeatedly calls the handle_client function to handle client requests.(It is for server need to ready to client everytime)
- Upon exiting the main loop (which should never happen as it's an infinite loop), it performs cleanup operations:
- Sends a SIGTERM signal to the process group to terminate all child processes.(Shouldn’t be reached here .)
- Detaches the shared memory segment from the process's address space.
- Marks the shared memory segment for deletion.
- Destroys the semaphore.

CLIENT

```c
int main(int argc, char *argv[])
{
    // check if the correct number of arguments
    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s <Connect/tryConnect> ServerPID\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *option = argv[1];// option: Connect or tryConnect
    int server_pid = atoi(argv[2]);// Server PID

    // create or get shared memory segment
    key_t key = ftok("server", 'R');
    int shm_id = shmget(key, SHM_SIZE, 0666);
    if (shm_id == -1)
    {
        perror("Failed to create shared memory segment");
        exit(EXIT_FAILURE);
    }

    // attach shared memory
    SharedData *shared_memory = (SharedData *)shmat(shm_id, NULL, 0);
    if (shared_memory == (void *)-1)
    {
        perror("Failed to attach shared memory");
        exit(EXIT_FAILURE);
    }
    if(server_pid!=shared_memory->client_pid){
        //printf("the pids are not match.")
        perror("The client pid is not match.");
        exit(EXIT_FAILURE);
    }
    // server PID to the shared memory
    shared_memory->client_pid=atoi(argv[2]);
    shared_memory->counter_client = shared_memory->counter_client +1 ; // increase counter
    if(shared_memory->counter_client==shared_memory->max_clients){//check queue
        perror("Queue is FULL MAX CLIENTS !\n");
        exit(EXIT_FAILURE);
    }
    if (strcmp(option, "Connect") == 0)
    {
        // Connect option
        printf("Connected to server with PID %d\n", shared_memory->client_pid);
        print_help();
        while (1)
        {
            printf("Enter command: ");
            // get command from user
            fgets(shared_memory->command, CMD_SIZE, stdin);
            //printf("%s", shared_memory->command);
            if (strcmp(shared_memory->command, "quit\n") == 0)
            {
                break;// exit loop if user enters 'quit'
            }
            // wait for server response and print it
            sleep(1);
            printf("Server response: %s\n", shared_memory->command);
        }
    }
    else if (strcmp(option, "tryConnect") == 0)
    {
        // tryConnect option
        printf("Connected to server with PID %d\n", shared_memory->client_pid);
        print_help();
    }
```

- The main function starts by checking if the correct number of command-line arguments is provided. It expects the program name, an option (Connect or tryConnect), and the server PID.

- Generates IPC key using ftok based on the file name "server" and the project identifier 'R'. Then, it creates or gets a shared memory segment using shmget with the generated key and specified size (SHM_SIZE).
- Attaches the shared memory segment to the address space of the calling process using shmat. If the attachment fails, it prints an error message and exits the program.
- Checks if the server PID matches the PID stored in the shared memory. If not, it prints an error message and exits the program.
- Updates the client PID and increments the client counter in the shared memory.
- Checks if the client queue is full. If so, it prints an error message and exits the program.
- Checks the provided option. If it's "Connect" or "tryConnect", it proceeds; otherwise, it prints an error message and exits.
- Detaches the shared memory segment from the process's address space using shmdt.

OUTPUTS/MAKEFILE/CHECK MEMORY LEAK

log.txt

```makefile
CC = gcc
CFLAGS = -Wall

all: clean compile run

compile: server client

server: server.c
	$(CC) $(CFLAGS) -o server server.c

client: client.c
	$(CC) $(CFLAGS) -o client client.c

run:
	@echo "Starting server..."
	@./server "/home/alper/Masaüstü/serverfile" 10 & echo $$! > server.pid
	@echo "Server started with PID `cat server.pid`"
	@./client Connect `cat server.pid`

clean:
	rm -f server client server.pid
```

alper@alper-VirtualBox: ~/Masaüstü/serverfile

```
alper@alper-VirtualBox:~/Masaüstü/serverfile$ make
rm -f server client server.pid
gcc -Wall  -o server server.c
gcc -Wall  -o client client.c
Starting server...
Server started PID 17081
Server started with PID 17081
Connected to server with PID 17081
Available commands are: help, list, readF, writeT, upload, download, archServer, killServer, quit
Enter command: help
Server response: Available commands are: help, list, readF, writeT, upload, download, archServer, kill
erver, quit
Enter command: list
Server response: server
<file>
makefile
server.c
log.txt
client.c
client
Ekran görüntüsü 2024-05-04 22-42-24.png
server.pid
Ekran görüntüsü 2024-05-04 22-42-20.png
Ekran görüntüsü 2024-05-04 22-42-25.png
Ekran görüntüsü 2024-05-04 22-42-18.png

Enter command: killServer
Kill signal received from client . Terminating...
Server response: killServer

Enter command: quit
alper@alper-VirtualBox:~/Masaüstü/serverfile$
```

alper@alper-VirtualBox: ~/Masaüstü/serverfile

```
lper@alper-VirtualBox:~/Masaüstü/serverfile$ valgrind --leak-check=full ./server
==16686== Memcheck, a memory error detector
==16686== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16686== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16686== Command: ./server
==16686==
Usage: ./server <dirname> <max. #ofClients>
==16686==
==16686== HEAP SUMMARY:
==16686==     in use at exit: 0 bytes in 0 blocks
==16686==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==16686==
==16686== All heap blocks were freed -- no leaks are possible
==16686==
==16686== For lists of detected and suppressed errors, rerun with: -s
==16686== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
lper@alper-VirtualBox:~/Masaüstü/serverfile$ valgrind --leak-check=full ./client
==16715== Memcheck, a memory error detector
==16715== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16715== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16715== Command: ./client
==16715==
Usage: ./client <Connect/tryConnect> ServerPID
==16715==
==16715== HEAP SUMMARY:
==16715==     in use at exit: 0 bytes in 0 blocks
==16715==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==16715==
==16715== All heap blocks were freed -- no leaks are possible
==16715==
==16715== For lists of detected and suppressed errors, rerun with: -s
==16715== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
lper@alper-VirtualBox:~/Masaüstü/serverfile$
```

```
1 Quitting serverAvailable commands are: help, list, readF, writeT, upload,
2 server.c
3 log.txt
4 client.c
5 client
6 Ekran görüntüsü 2024-05-04 22-42-24.png
7 Ekran görüntüsü 2024-05-04 22-42-20.png
8 Ekran görüntüsü 2024-05-04 22-42-25.png
9 Ekran görüntüsü 2024-05-04 22-42-18.png
10 Available commands are: help, list, readF, writeT, upload, download, archS
   quitAvailable commands are: help, list, readF, writeT, upload, download, a
   quitserver
11 server.c
12 log.txt
13 client.c
14 client
15 Ekran görüntüsü 2024-05-04 22-42-24.png
16 Ekran görüntüsü 2024-05-04 22-42-20.png
17 Ekran görüntüsü 2024-05-04 22-42-25.png
18 Ekran görüntüsü 2024-05-04 22-42-18.png
19 Available commands are: help, list, readF, writeT, upload, download, archS
20 server.c
21 log.txt
22 client.c
23 client
24 clien
25 Ekran görüntüsü 2024-05-04 22-42-24.png
26 Ekran görüntüsü 2024-05-04 22-42-20.png
27 Ekran görüntüsü 2024-05-04 22-42-25.png
28 Ekran görüntüsü 2024-05-04 22-42-18.png
29 Quitting serverAvailable commands are: help, list, readF, writeT, upload,
   killServer, quitAvailable commands are: help, list, readF, writeT, upload,
   killServer, quitAvailable commands are: help, list, readF, writeT, upload,
   readF, writeT, upload, download, archServer, killServer, quitAvailable com
30 server.c
31 log.txt
32 client.c
33 client
34 Ekran görüntüsü 2024-05-04 22-42-24.png
35 Ekran görüntüsü 2024-05-04 22-42-20.png
36 Ekran görüntüsü 2024-05-04 22-42-25.png
37 Ekran görüntüsü 2024-05-04 22-42-18.png
38 readF <file> <line #>
39 readF <file> <line #>
40 Available commands are: help, list, readF, writeT, upload, download, archS
41 server.c
42 log.txt
43 client.c
44 client
45 Ekran görüntüsü 2024-05-04 22-42-24.png
```

```
alper@alper-VirtualBox: ~/Masaüstü/serverfile
alper@alper-VirtualBox:~/Masaüstü/serverfile$ gcc client.c -o client
alper@alper-VirtualBox:~/Masaüstü/serverfile$ ./client Connect 13560
The client pid is not match.: Success
alper@alper-VirtualBox:~/Masaüstü/serverfile$ ./client Connect 13954
Connected to server with PID 13954
Available commands are: help, list, readF, writeT, upload, download, archServer
, killServer, quit
Enter command: help
Server response: Available commands are: help, list, readF, writeT, upload, dow
nload, archServer, killServer, quit
Enter command: list
Server response: server
server.c
log.txt
client.c
client
Ekran görüntüsü 2024-05-04 22-42-24.png
Ekran görüntüsü 2024-05-04 22-42-20.png
Ekran görüntüsü 2024-05-04 22-42-25.png
Ekran görüntüsü 2024-05-04 22-42-18.png

Enter command: help readF
Server response: readF <file> <line #>

Enter command: readF server.c 10
Server response: #include <fcntl.h>

Enter command: readF log.txt10
Server response:
Enter command: readF log.txt 10
Server response: Available commands are: help, list, readF, writeT, upload, dow
nload, archServer, killServer, quitAvailable commands are: help, list, readF, w
riteT, upload, download, archServer, killServer, quitAvailable commands are: he
lp, list, readF, writeT, upload, do
Enter command: help writeT
Server response: writeT <file> <line #> <string>

Enter command: writeT log.txt 10  "Alper Uzuntepe"
```

download, archServer, killServer,
oad, download, archServer, killServer,
download, archServer, quitserver
writeT, upload, download, archServer,
writeT, upload, download, archServer,
erAvailable commands are: help, list,
uitserver

```
alper@alper-VirtualBox: ~/Masaüstü/serverfile
alper@alper-VirtualBox:~/Masaüstü/serverfile$ ./client Connect 15374
Connected to server with PID 15374
Available commands are: help, list, readF, writeT, upload, download, archServe
r, killServer, quit
Enter command: help upload
Server response: upload <file>

Enter command: upload log.txt
Server response: File uploaded successfully
Enter command: download log.txt
Server response: File downloaded successfully
Enter command: upload log.txt
Server response: File uploaded successfully
Enter command: download hw2.c
Server response: Failed to open source file
Enter command: download HW2.c
Server response: File downloaded successfully
Enter command: help ArchServer
Server response:
Enter command: help archServer
Server response: archServer <fileName>.tar

Enter command: archServer deneme.tar
Server response: archServer deneme.tar

Enter command: help
Server response: Available commands are: help, list, readF, writeT, upload, do
wnload, archServer, killServer, quit
Enter command: killServer
Server response: killServer

Enter command: quit
alper@alper-VirtualBox:~/Masaüstü/serverfile$
```

```
alper@alper-VirtualBox: ~/Masaüstü/serverfile
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/glean/db/data.saf
e.bin
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/session-state.jso
n
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/2024-03/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/2024-03/
1710801235444.490a8f93-1117-4dac-bf75-cef54ad61008.main.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/2024-03/
1710801235411.00cd943c-c1e4-4b3a-803f-e548011b5000.health.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/2024-03/
1710801235263.e23dd600-ec94-4731-a791-2ec1e8c184d4.health.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/archived/2024-03/
1710801235407.338338ed-710d-41c7-a732-3abef9132d55.event.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/datareporting/state.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/protections.sqlite
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/activity-stream.discovery_strea
m.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/minidumps/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/broadcast-listeners.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/xulstore.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionstore-backups/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionstore-backups/upgrade.js
onlz4-20230406114409
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionstore-backups/previous.j
sonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionstore-backups/recovery.b
aklz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionstore-backups/recovery.j
sonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/ExperimentStoreData.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/features/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/features/{95ade27d-59ff-4e8d-82
27-695d7748fc7d}/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/features/{95ade27d-59ff-4e8d-82
27-695d7748fc7d}/addons-restricted-domains@mozilla.com.xpi
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/serviceworker.txt
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/.parentlock
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/times.json
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/permissions.sqlite
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/addonStartup.json.lz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/extensions/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/pkcs11.txt
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/bookmarkbackups/
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/bookmarkbackups/bookmarks-2022-
04-15_16_IalZZndSClOjb5KqcHcVxw==.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/bookmarkbackups/bookmarks-2023-
05-18_16_zMpcPsQP2XWlpdVOElI1hA==.jsonlz4
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/credentialstate.sqlite
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/compatibility.ini
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/cert9.db
/home/alper/MasaüstÜ/Old Firefox Data/wbvs1sn5.default-release/sessionCheckpoints.json
Kill signal received from client . Terminating...
alper@alper-VirtualBox:~/Masaüstü/serverfile$
```