CSE 344

SYSTEM PROGRAMMING

SPRING 2023

FINAL

AHMET CEBECİ

1901042708

INTRODUCTION

The provided code consists of two programs: a server and a client, designed to facilitate the transfer of files from the server to the client. The server program listens for incoming connections and sends files to connected clients. The client program connects to the server and receives files sent by the server. In this report, we will discuss how the code functions and provide an overview of its main components.

SERVER

```
void handle_sigint(int sig) {
    printf("Received SIGINT. Shutting down server.\n");
    exit(0);
}
```

Handling SIGINT:

The server program begins by defining a signal handler function handle_sigint(), which is used to handle the SIGINT signal (generated by pressing Ctrl+C).

When SIGINT is received, the function prints a message and terminates the server gracefully by calling exit(0).

```c
void send_file(int client_socket, char *file_name) {
    FILE *file = fopen(file_name, "rb");
    if (file == NULL) {
        perror("File open error");
        return;
    }

    char file_name_buffer[256];
    strncpy(file_name_buffer, file_name, sizeof(file_name_buffer));
    send(client_socket, file_name_buffer, sizeof(file_name_buffer), 0);

    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);
    send(client_socket, &file_size, sizeof(file_size), 0);

    char buffer[256];
    size_t n;
    while ((n = fread(buffer, sizeof(char), sizeof(buffer), file)) > 0) {
        if (send(client_socket, buffer, n, 0) < 0) {
            perror("Failed to send file");
            fclose(file);
            return;
        }
    }

    fclose(file);
}
```

Sending Files:

The send_file() function is responsible for sending an individual file to the client. It takes the client socket and the name of the file as input.

It opens the file in binary mode, checks for any errors in opening the file, and proceeds to send the file's name and size to the client using the send() function.

The function then reads the file in chunks and sends each chunk to the client until the entire file has been sent.

Finally, the file is closed.

```c
void send_directory_contents(int client_socket, const char *directory) {
    DIR *dir;
    struct dirent *entry;
    struct stat file_stat;

    if ((dir = opendir(directory)) == NULL) {
        perror("Failed to open directory");
        return;
    }

    chdir(directory);

    while ((entry = readdir(dir)) != NULL) {
        stat(entry->d_name, &file_stat);
        if (S_ISREG(file_stat.st_mode)) {
            printf("Sending file: %s\n", entry->d_name);
            send_file(client_socket, entry->d_name);
        }
    }

    chdir("..");
    closedir(dir);
}
```

Sending Directory Contents:

The send_directory_contents() function is responsible for sending all the regular files present in a specified directory to the client.

It opens the specified directory using opendir() and iterates over each entry using readdir().

For each regular file encountered, it calls the send_file() function to send the file to the client.

```c
void *client_handler(void *socket_desc) {
    int client_socket = *(int *)socket_desc;
    send_directory_contents(client_socket, directory);
    close(client_socket);
    free(socket_desc);
    return 0;
}
```

Client Handler:

The client_handler() function is the entry point for each client thread.

It takes the client socket as input, calls send_directory_contents() to send the directory contents to the client, and then closes the client socket.

After closing the socket, it frees the memory allocated for the client socket descriptor.

```c
int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Usage: ./server <directory> <threadPoolSize> <portnumber>\n");
        return 1;
    }

    directory = argv[1];
    int threadPoolSize = atoi(argv[2]);
    int portnumber = atoi(argv[3]);

    struct stat st = {0};

    if (stat(directory, &st) == -1) {
        mkdir(directory, 0777);
    }

    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;

    server_socket = socket(AF_INET, SOCK_STREAM, 0);

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(portnumber);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    bind(server_socket, (struct sockaddr *) &server_addr, sizeof(server_addr));
    listen(server_socket, threadPoolSize);

    signal(SIGINT, handle_sigint);

    printf("Server is listening on port %d\n", portnumber);

    pthread_t thread_id;
    while (1) {
        addr_size = sizeof(client_addr);
        client_socket = accept(server_socket, (struct sockaddr *) &client_addr, &addr_size);
        printf("Connection accepted\n");

        int *new_sock;
        new_sock = malloc(sizeof(int));
        *new_sock = client_socket;

        pthread_create(&thread_id, NULL, client_handler, (void *)new_sock);
    }

    close(server_socket);

    return 0;
}
```

Main Function:

The main() function of the server program validates the command-line arguments, which should include the directory path, thread pool size, and port number.

It creates the specified directory if it does not already exist using mkdir().

The server socket is created using socket() and is bound to the specified port number using bind().

The server socket starts listening for incoming connections using listen().

A signal handler is set up to handle the SIGINT signal using signal().

The main loop accepts incoming client connections using accept(), creates a new thread for each connection using pthread_create(), and repeats the process indefinitely.

Finally, the server socket is closed.

CLIENT

```
void handle_sigint(int sig) {
    printf("Received SIGINT. Closing client.\n");
    exit(0);
}
```

Handling SIGINT:

The client program begins by defining a signal handler function handle_sigint(), which is used to handle the SIGINT signal (generated by pressing Ctrl+C).

When SIGINT is received, the function prints a message and terminates the client program gracefully by calling exit(0).

```c
int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: ./client <directory> <portnumber>\n");
        return 1;
    }

    char *dirName = argv[1];
    int portnumber = atoi(argv[2]);

    struct stat st = {0};

    if (stat(dirName, &st) == -1) {
        if (mkdir(dirName, 0700) != 0) {
            perror("Directory creation error");
            return 1;
        }
    } else {
        printf("Directory %s already exists\n", dirName);
    }
    if (chdir(dirName) != 0) {
        perror("Failed to change directory");
        return 1;
    }

    signal(SIGINT, handle_sigint);

    while(1) {
        int client_socket;
        struct sockaddr_in server_addr;
        client_socket = socket(AF_INET, SOCK_STREAM, 0);

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(portnumber);
        server_addr.sin_addr.s_addr = INADDR_ANY;

        if (connect(client_socket, (struct sockaddr *) &server_addr, sizeof(server_addr)) < 0) {
            perror("Connection error");
            close(client_socket);
            sleep(10);
            continue;
        }

        printf("Connected to server\n");

        char file_name[256];
        long file_size;
        char buffer[256];

        while (1) {
            if (recv(client_socket, file_name, sizeof(file_name), 0) <= 0) break;
            if (recv(client_socket, &file_size, sizeof(file_size), 0) <= 0) break;
            FILE *file = fopen(file_name, "wb");
            if (file == NULL) {
                perror("Error opening file for writing");
                continue;
            }
            long remaining_bytes = file_size;
            while (remaining_bytes > 0) {
                size_t n = recv(client_socket, buffer, sizeof(buffer), 0);
                if (n <= 0) break;
                fwrite(buffer, sizeof(char), n, file);
                remaining_bytes -= n;
            }

            fclose(file);
            printf("File received: %s\n", file_name);
        }

        close(client_socket);
        printf("Connection closed. Reconnecting in 30 seconds...\n");

        sleep(10);
```

Connecting to the Server:

The client program connects to the server using the provided directory path and port number.

It repeatedly attempts to connect until a successful connection is established, sleeping for 10 seconds between each attempt.

Receiving Files:

The client program receives files sent by the server in a loop.

It receives the file name and size using the recv() function.

It then opens the file for writing in binary mode, reads data from the server in chunks using recv(), and writes the received data to the file until the entire file has been received.

Finally, the file is closed, and a message is printed indicating that the file has been received.

Main Function:

The main() function of the client program validates the command-line arguments, which should include the local directory path and the server's port number.

It creates the specified directory if it does not already exist using mkdir().

The SIGINT signal handler is set up using signal().

The client program enters a loop, attempting to connect to the server, receiving files, and closing the connection after each file has been received.

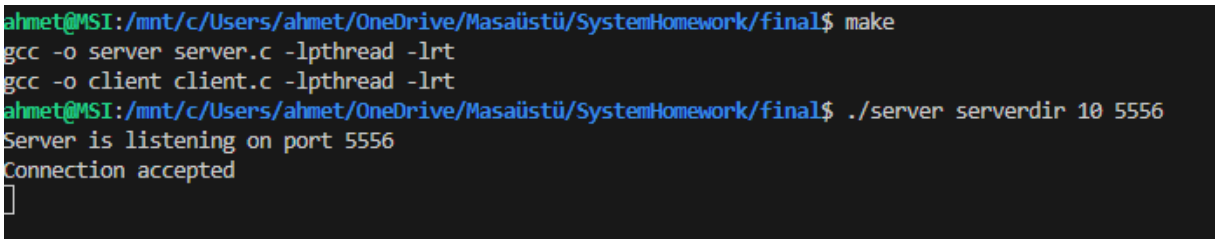
After the connection is closed, a message is printed indicating that the connection has been closed, and the client program waits for 10 seconds before attempting to reconnect.

SUMMARY:

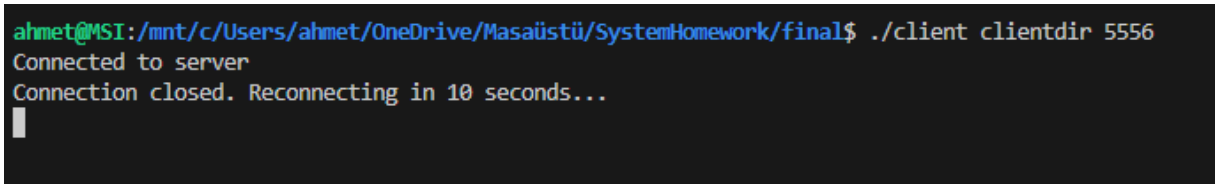
The provided server and client code allows for the transfer of files from the server to the client. The server program listens for incoming connections, sends the contents of a specified directory to each connected client, and terminates gracefully upon receiving a SIGINT signal. The client program connects to the server, receives files sent by the server, and attempts to reconnect to the server in case of disconnection. Both programs handle errors related to file operations, socket connections, and directory manipulation.

OUTPUT

Server:



Client:



After adding a file in server

Server:

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework/final$ ./server serverdir 10 5555
Server is listening on port 5555
Connection accepted
Sending file: final_project.docx
```

Client:

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework/final$ ./client clientdir 5555
Connected to server
File received: final_project.docx
Connection closed. Reconnecting in 10 seconds...
```

Another copy test

Server:

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework/final$ ./server serverdir 10 5556
Server is listening on port 5556
Connection accepted
Sending file: AHMET CEBECİ GEBZE TEKNİK ÜNİVERSİTESİ (1).pdf
Sending file: final_project.docx
Sending file: FR-0075_Staj_Yeri_Kabul_Belgesi (1) (1).xls
Sending file: image (21).txt
```

Client:

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework/final$ ./client clientdir 5556
Connected to server
File received: AHMET CEBECİ GEBZE TEKNİK ÜNİVERSİTESİ (1).pdf
File received: final_project.docx
File received: FR-0075_Staj_Yeri_Kabul_Belgesi (1) (1).xls
File received: image (21).txt
Connection closed. Reconnecting in 10 seconds...
```