National University of Sciences & Technology

School of Electrical Engineering and Computer Science

Department of Computing

EE353 & EE347: Computer Network (3+1): BSCS-12(C) & BSDS-1, Fall 2024

Name	**Qalam**
Abdul Mateen	**457052**
Muhammad Usman Naeem	**481453**

Reciver.c

```
#include <arpa/inet.h>

// This header file provides definitions for internet operations, such as converting IP addresses to a format suitable for socket communication.

#include <stdio.h>

// Standard input/output library. Provides functions like printf() for printing output and fopen() for file handling.

#include <stdlib.h>

// Provides standard utility functions such as exit() and memory management functions like malloc().

#include <string.h>

// Provides functions for handling strings, like memset() and strcmp().

#include <sys/socket.h>

// Contains definitions for socket operations, like creating a socket, connecting, sending, and receiving data.

#include <unistd.h>

// Provides access to the POSIX operating system API, including functions for file operations (close()) and other system calls.

#include <netinet/in.h>

/*This header file is part of the Berkeley sockets API and provides constants, structures,

and functions for manipulating addresses and sockets in the IPv4 and IPv6 Internet protocol family. */

#include <stdbool.h>

/*This is a standard C library introduced in C99 that allows the use of Boolean data types (true/false) in C programs.
```

```
Before stdbool.h, C did not have a native bool type, and programmers used integers (0 for false,
non-zero for true).*/


#define CHUNK_SIZE 1024

// CHUNK_SIZE is a macro that defines the size of each chunk of data (1024 bytes) to be sent
over the network in one transmission.


int main() {

int SERVER_PORT = 8877;

socklen_t client_address_len;


// Setup server address


struct sockaddr_in server_address; // This structure holds the address of the server. It contains the
IP address and the port number.

memset(&server_address, 0, sizeof(server_address)); // this clears the memory of the
server_address

server_address.sin_port = htons(SERVER_PORT); // This converts the port number server_port
(8877) into network byte order using htons().

server_address.sin_family = AF_INET; // it indicates the use of IPv4 addresses

server_address.sin_addr.s_addr = htonl(INADDR_ANY);


// Create socket

int listen_sock;

if ((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {

printf("Could not create listen socket\n");

return 1;

}


// Bind socket

if ((bind(listen_sock, (struct sockaddr *)&server_address, sizeof(server_address))) < 0) {

printf("Could not bind socket\n");

return 1;
```

```
}
else{
printf("Connected with the sender\n");
}

// Listen for incoming connections
int wait_size = 16;
if (listen(listen_sock, wait_size) < 0) {
printf("Could not open socket for listening\n");
return 1;
}

// Accept connections and process the file transmission
while (true) {
// Accept a connection
struct sockaddr_in client_address;
int sock;
if ((sock = accept(listen_sock, (struct sockaddr*)&client_address, &client_address_len)) < 0) {
printf("Could not open socket to accept data\n");
return 1;
}

// Corrected: inet_ntoa expects `client_address.sin_addr` as the argument
printf("Client connected with IP address: %s\n", inet_ntoa(client_address.sin_addr));

// Open a new file to save the received video
FILE* file = fopen("video.mp4", "wb"); // Path for saving the received video
if (!file) {
printf("Could not open file to write\n");
return 1;
}
```

```
else{
printf("File is being recieved\n");
}

// Receive data in chunks and write to the file
char buffer[CHUNK_SIZE];
int n;
while ((n = recv(sock, buffer, CHUNK_SIZE, 0)) > 0) {
if (strncmp(buffer, "EOF", 3) == 0) { // Check for EOF marker
break;
}
fwrite(buffer, 1, n, file);
}

// Close the file and socket
fclose(file);
close(sock);

printf("File received successfully\n");
}

close(listen_sock);
return 0;
}
```

Server.c

```
#include <arpa/inet.h>
// This header file provides definitions for internet operations, such as converting IP addresses to a format suitable for socket communication.
```

```
#include <stdio.h>

// Standard input/output library. Provides functions like printf() for printing output and fopen() for
file handling.

#include <stdlib.h>

// Provides standard utility functions such as exit() and memory management functions like
malloc().

#include <string.h>

// Provides functions for handling strings, like memset() and strcmp().

#include <sys/socket.h>

// Contains definitions for socket operations, like creating a socket, connecting, sending, and
receiving data.

#include <unistd.h>

// Provides access to the POSIX operating system API, including functions for file operations
(close()) and other system calls.


#define CHUNK_SIZE 1024

// CHUNK_SIZE is a macro that defines the size of each chunk of data (1024 bytes) to be sent
over the network in one transmission.


int main() {

const char* server_name = "Abdul Mateen";

const int server_port = 8877; //The port number (8877) that the server is listening on for incoming
connections.


// Set up server address

struct sockaddr_in server_address; // This structure holds the address of the server. It contains the
IP address and the port number.

memset(&server_address, 0, sizeof(server_address)); // this clears the memory of the
server_address

server_address.sin_family = AF_INET; // it indicates the use of IPv4 addresses

inet_pton(AF_INET, server_name, &server_address.sin_addr); // This converts the string
server_name into a binary format

server_address.sin_port = htons(server_port); // This converts the port number server_port (8877)
into network byte order using htons().
```

```
// Create socket
int sock;
if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
printf("Could not create socket\n");
return 1;
}

// Connect to server
if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
printf("Could not connect to reciever\n");
printf("Trying to connect to %s:%d\n", server_name, server_port);

return 1;
}
else{
printf("Connecting to the reciever\n");
}

// Open video file
FILE* file = fopen("video.mp4", "rb"); // Replace with your video file path
if (!file) {
printf("Could not open video file\n");
return 1;
}
else{
printf("File is ready to be sended\n");
}

// Send file contents in chunks
char buffer[CHUNK_SIZE];
```

```
size_t bytes_read;

while ((bytes_read = fread(buffer, 1, CHUNK_SIZE, file)) > 0) {

send(sock, buffer, bytes_read, 0);

}


// Send EOF marker (e.g., an empty message or a special character sequence)

send(sock, "EOF", 3, 0); // You can change the EOF marker if needed


// Close the file and socket

fclose(file);

close(sock);


printf("File sent successfully\n");

return 0;

}
```

Video Link:
https://github.com/AbdulMateen12344567/sender-to-reciever-/blob/main/demovideo.webm