AUDIO AND VIDEO FILE TRANSFER USING UDP

MINI PROJECT REPORT

Submitted by

ARUNKUMAR K(202209006)

GOKUL ANAND P(202209013)

KANAGA BALA R(202209023)

in

19AD551 – COMPUTER NETWORKING LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

NOVEMBER 2024

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **"ARUNKUMAR K (202209006), GOKULANAND P(202209013),KANAGA BALA R(202209023)"** for the mini project titled **"AUDIO AND VIDEO FILE TRANSFER USING UDP"** in 19AD551 – Computer Networking Laboratory during the fifth semester July 2024 – November 2024 under my supervision.

SIGNATURE

Dr. P. Swathika,
Asst. Professor (Senior Grade),
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Professor & Head,
AI&DS Department
Mepco Schlenk Engg. College, Sivakasi

ABSTRACT

This project aims to develop a UDP file transfer application that facilitates efficient data exchange between a client and a server over a local network. By leveraging socket programming in C, students will gain foundational knowledge of network communications, including the creation and management of sockets. The application will enable users to request specific files from the server, which will be sent in manageable data chunks, ensuring a reliable transfer process. Additionally, the project will illustrate key file I/O operations, utilizing system calls to read from and write to files effectively. Participants will also explore the advantages and limitations of the User Datagram Protocol (UDP) in comparison to other protocols such as TCP. The project intends to create a user-friendly interface for file requests and saves, while implementing basic error handling to manage common communication issues. Although primarily focused on basic functionality, the design allows for future enhancements, such as support for multiple simultaneous clients and integration of additional protocols, making this project a stepping stone toward more complex networking applications.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

In today's digital age, efficient data transfer over networks is paramount for business operations, communication, and various applications. With the exponential rise in remote work and the demand for real-time collaboration, understanding how data packets traverse networks has become increasingly important. This project presents a user-friendly UDP file transfer application implemented in the C programming language, aimed at facilitating the seamless retrieval of files over a local area network.

The application consists of two components are a client that requests files and a server that handles these requests and sends back the appropriate file data. Implementing this project provided insights into socket programming, including how to create sockets, bind them to ports, transmit data over UDP, and manage file input/output operations in a Unix/Linux environment. The simplicity and speed of UDP make it a suitable choice for applications where real-time performance is critical, despite its lack of reliability compared to TCP.

1.2 OBJECTIVES

1.2.1 Understanding Socket Programming:

The fundamental aim of this project is to develop a solid understanding of socket programming in the C programming language. Students will learn about the core concepts associated with socket creation, including how to establish connections between clients and servers. They will be introduced to different types of sockets, specifically stream sockets for TCP and datagram sockets for UDP. Additionally, the project will cover the essential API functions used in socket programming, enabling students to effectively manage network communications. By engaging with hands-on coding exercises, participants will gain practical knowledge that can be applied to real-world networking challenges.

1.2.2 Building a UDP File Transfer Application:

The project will involve the construction of a file transfer application based on the User Datagram Protocol (UDP). Students will be tasked with designing a reliable client-server architecture that allows clients to request specific files from the server. The implementation will focus on efficiently transferring files in manageable blocks that ensure data integrity and reliability during the transfer. Key considerations will include developing mechanisms to acknowledge receipt of file chunks and handle packet loss, which is inherent in UDP communications. By the end of this objective, participants should be able to produce a functioning application that can effectively transfer files across a network.

1.2.3 Demonstrating File I/O Operations:

Another critical objective is to illustrate various file input/output (I/O) operations using system calls available in C programming. Participants will learn how to open, read, write, and close files through standard C libraries and system calls, which are essential for managing data on a filesystem. This knowledge is vital for implementing the file transfer operations required by the project. Students will also experiment with different file types and scenarios, reinforcing their understanding of file management within the context of a networked application. Learning to handle files in a C environment will provide students with skills relevant to many programming applications.

1.2.4 Investigating Networking Protocols:

As part of this project, students will investigate the User Datagram Protocol (UDP) and how it differs from other networking protocols, particularly Transmission Control Protocol (TCP). They will analyze the advantages of using UDP, such as its lower latency and suitability for applications requiring fast, efficient data transmission. However, they will also examine the drawbacks of using UDP, such as the lack of guaranteed delivery and the possibilities of packet

loss. This comparative analysis will help students understand the implications of protocol selection in application development and enhance their ability to make informed decisions when designing networked systems.

1.2.5 Handling User Input:

A user-friendly interface is crucial for effective client-server communication, and this project will emphasize the importance of handling user input effectively. Participants will design an interface that allows users to specify the file they wish to request from the server and determine the desired location for saving the transferred file on their local machine. This component will require implementing input validation techniques to ensure that users' input meets the necessary criteria for successful file requests and storage. By focusing on usability, students will learn how to create more engaging and accessible applications that prioritize user experience in their designs.

1.3 SCOPE OF PROJECT

1.3.1 Basic File Transfer Functionality:

The project will primarily focus on implementing basic file transfer functionality, where clients can request files that are readily available on the server. The server will be programmed to provide requested files in manageable chunks, enabling easier handling of data transmission and minimizing the risks associated with large file transfers. The core aim is to achieve a working prototype that emphasizes simplicity and reliability in transferring files over the network. Participants will concentrate on ensuring that the basic transfer operations are functional before considering more advanced features. This foundational implementation will serve as the backbone for potential future enhancements.

1.3.2 Local Networking:

To streamline testing and demonstration, the project will operate primarily over a local network, utilizing the loopback IP address (127.0.0.1). This approach allows students to focus on developing and debugging their applications in a controlled environment, free from the complexities introduced by external networking configurations. Using local networking will enable quicker iterations in development and facilitate hands-on experimentation with socket programming concepts. Participants will gain experience managing local network operations, which will build their confidence in taking their projects to more extensive networking scenarios in the future.

1.3.3 File Formats and Types:

While the application will be designed to handle a variety of file formats, the focus during initial testing will be on text files and small media files. This approach simplifies testing, validation, and debugging processes, allowing students to familiarize themselves with file transfer operations without the added challenges of handling larger or more complex file types. By working with these manageable file types, participants can efficiently test the application's functionality and ensure that the transfer process is reliable before moving on to more sophisticated or larger file formats in the later phases of the project.

1.3.4 Error Handling:

An essential aspect of any application is effective error handling, and while this project will include basic error management capabilities, extensive error handling is outside its current scope. Participants will implement fundamental error checks to ensure that users are informed of any issues related to file access or communication failures during program execution. By addressing common potential errors, such as file not found or network connectivity issues, students will learn valuable strategies for providing feedback to users and maintaining application stability.

The focus will remain on achieving a functioning prototype, with advanced error handling considered for future iterations.

1.3.4 Potential Extensions:

While the current implementation is designed to support a single client at a time, there are several potential extensions that could enhance the application's functionality. Future versions could explore providing support for multiple concurrent clients, allowing the server to handle multiple file transfer requests simultaneously. Additionally, integrating other protocols like TCP could add features such as secure transmission and reliable delivery. Participants may also consider improving the user interface for better usability and feedback. These potential extensions will encourage students to think critically about scalability, performance, and the user experience as they advance their projects.

CHAPTER-2

IMPLEMENTATION

2.1 PROTOCOLS USED

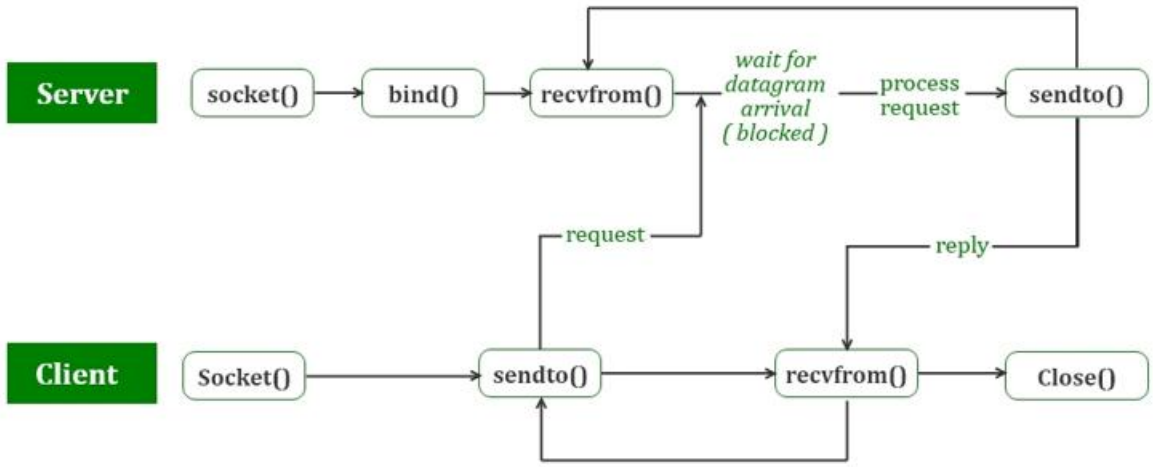

Figure 2.1: User Datagram Protocol Architecture

User Datagram Protocol is a connectionless, unreliable transport protocol that lies between the application and transport layers. It does not add anything to IP services except for providing a process to process communication instead of the host to host evidence.User Datagram Protocol transfers data in packet format, which has 8 bytes of the header that includes parameters like source port number, destination port number, total length, checksum each of 16 bits.UDP provides services like the process to process communication, connectionless services, flow control, error control, checksum, congestion control, encapsulation, and decapsulation, queuing, multiplexing, and demultiplexing.

2.2 PROGRAM CODING

2.2.1 Client code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <unistd.h>
#include <arpa/inet.h>

void set_socket(struct sockaddr_in *socket, int type, int host_short) {
   socket->sin_family = type;
   socket->sin_port = htons(host_short);
}

int main() {
   int sid = 0;
   char *send_data = (char *)malloc(1024);
   char *receive_data = (char *)malloc(1024);
   struct sockaddr_in server_socket;
   socklen_t size = sizeof(server_socket);

   if ((sid = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
```

```
        perror("Connection error at client side");
        exit(1);
    }

    set_socket(&server_socket, AF_INET, 6000);

    if (inet_aton("127.0.0.1", &server_socket.sin_addr) == 0) {
        fprintf(stderr, "inet_aton() failed\n");
        free(send_data);
        free(receive_data);
        close(sid);
        exit(1);
    }

    // Additional input for output filename
    char output_filename[256];
    printf("Enter the name under which to save the file: ");
    scanf("%s", output_filename);

    // Request from user for the file name to send to server
    printf("Enter the name of the file you want to request: ");
    scanf("%s", send_data);

    // Send the filename request to the server
    sendto(sid, send_data, strlen(send_data), 0, (struct sockaddr *)&server_socket,
size);
```

```
    // Open the file to write received data
    int fd = open(output_filename, O_CREAT | O_WRONLY | O_TRUNC,
S_IRUSR | S_IWUSR | S_IXUSR);
    if (fd == -1) {
        perror("File creation error");
        free(send_data);
        free(receive_data);
        close(sid);
        return 1;
    }

    // Receive data from the server
    while (1) {
        int bytes_received = recvfrom(sid, receive_data, 1024, 0, (struct sockaddr
*)&server_socket, &size);

        if (bytes_received < 0) {
            perror("Receive error");
            break;
        }

        if (strncmp(receive_data, "ENDOFFILE", 9) == 0) {
            printf("End of file transmission.\n");
            break;
        } else {
            // Write the received data directly to the file
            write(fd, receive_data, bytes_received);
```

```
        }
    }

    close(fd);
    close(sid);
    free(send_data);
    free(receive_data);

    return 0;
}
```

2.2.2 Server code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <unistd.h>
#include <arpa/inet.h>

void set_socket(struct sockaddr_in *socket, int type, int host_short) {
    socket->sin_family = type;
    socket->sin_port = htons(host_short);
    socket->sin_addr.s_addr = htonl(INADDR_ANY);
```

```
}

int main() {
    int sid = 0, fd;
    char *temp = (char *)malloc(500); // Buffer for reading the file
    char *receive_data = (char *)malloc(1024);
    struct sockaddr_in server_socket, client_socket;
    socklen_t size = sizeof(client_socket);

    if ((sid = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        perror("Socket creation error");
        free(temp);
        free(receive_data);
        exit(1);
    }

    set_socket(&server_socket, AF_INET, 6000);

    if (bind(sid, (struct sockaddr *)&server_socket, sizeof(server_socket)) == -1) {
        perror("Binding error");
        free(temp);
        free(receive_data);
        close(sid);
        exit(1);
    }

    printf("Server is waiting for client...\n");
```

```
    // Receive the filename from the client
    recvfrom(sid, receive_data, 1024, 0, (struct sockaddr *)&client_socket, &size);
    printf("Received request for file: %s\n", receive_data);
    fd = open(receive_data, O_RDONLY);

    if (fd == -1) {
        perror("File open error");
        free(temp);
        free(receive_data);
        close(sid);
        return 1;
    }

    // Send file data to client
    int count;
    while ((count = read(fd, temp, 500)) > 0) {
        printf("Sending data chunk of size %d...\n", count);
        sendto(sid, temp, count, 0, (struct sockaddr *)&client_socket, size);
    }

    // Indicate end of file transfer
    const char *end_signal = "ENDOFFILE";
    sendto(sid, end_signal, strlen(end_signal), 0, (struct sockaddr *)&client_socket,
size);

    printf("File transfer complete.\n");
```

```
    close(fd);
    close(sid);
    free(temp);
    free(receive_data);
    return 0;
}
```

2.3 OUTPUT:

2.3.1 Server output:

```
gokul@DESKTOP-NNALQ9T:/mnt/c/Users/ASUS/Music/Music_1$ ./a.out
Server is waiting for client...
```

Figure2.3.1:Server listening

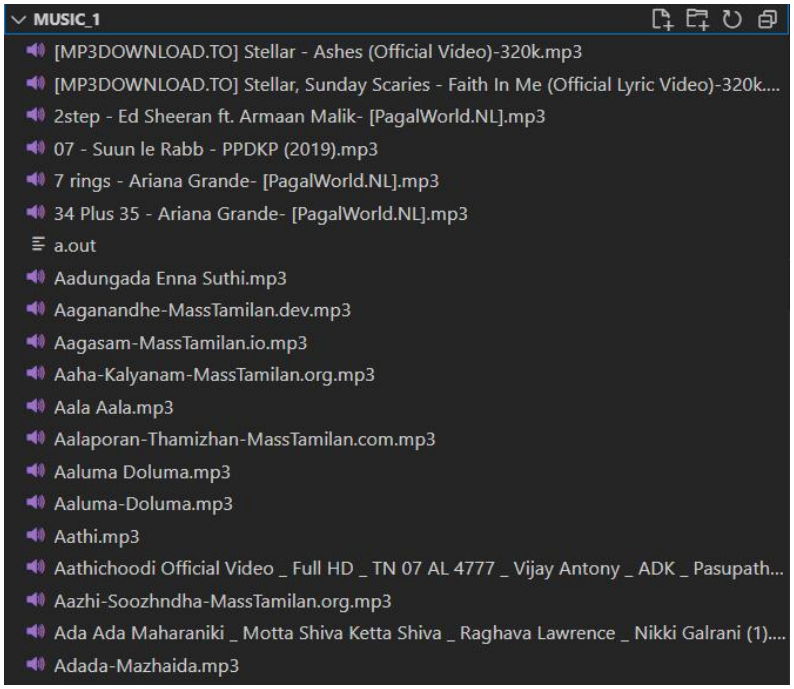

Figure2.3.1:Server directory files

```
gokul@DESKTOP-NNALQ9T:/mnt/c/Users/ASUS/Music$ cc client.c
gokul@DESKTOP-NNALQ9T:/mnt/c/Users/ASUS/Music$ ./a.out
Enter the name under which to save the file: abc.mp3
Enter the name of the file you want to request: Aaluma-Doluma.mp3
End of file transmission.
```

Figure2.3.1:Server transmission success

2.2.3 Client output:

```
gokul@DESKTOP-NNALQ9T:/mnt/c/Users/ASUS/Music$ cc client.c
gokul@DESKTOP-NNALQ9T:/mnt/c/Users/ASUS/Music$ ./a.out
Enter the name under which to save the file: abc.mp3
Enter the name of the file you want to request: Aaluma-Doluma.mp3
```

Figure2.3.3:Client request files

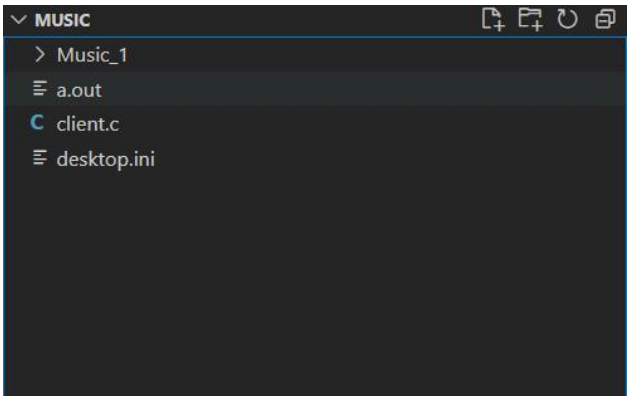

Figure2.3.3:Client directory before transfer

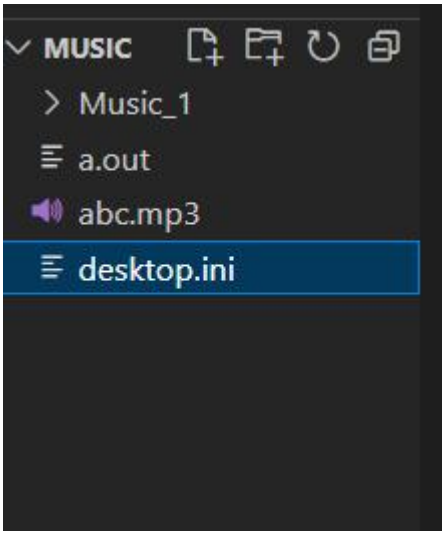

Figure2.3.3:Client directory after transfer

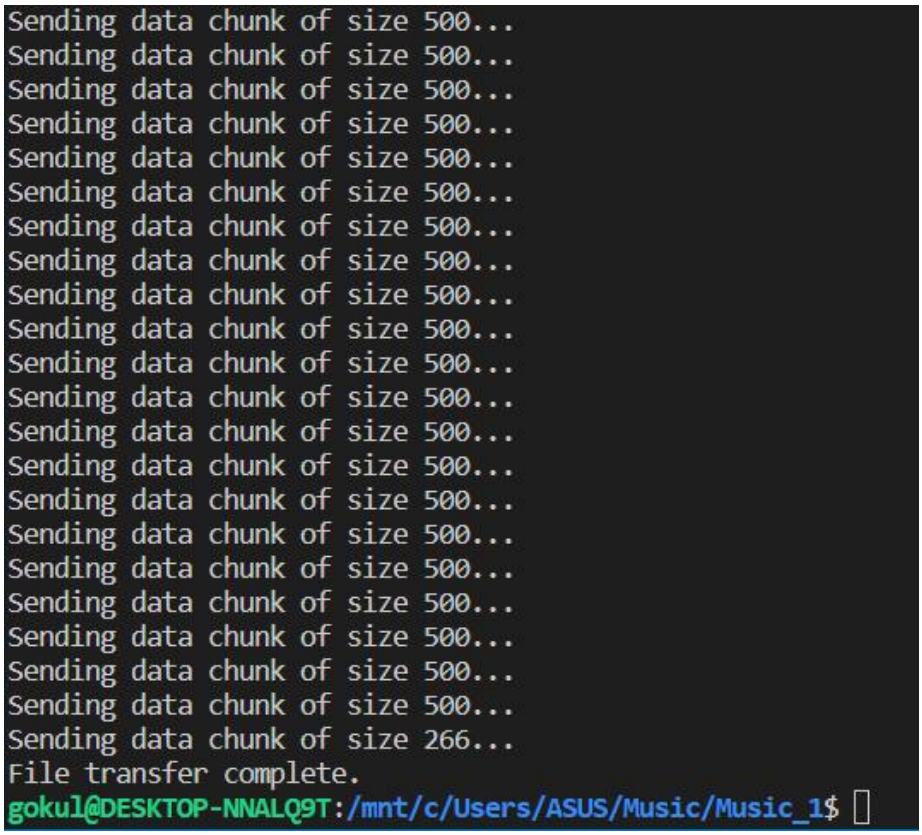

Figure2.3.3: Transfered successfully

CHAPTER-3
CONCLUSION

The UDP file transfer application successfully demonstrates the fundamental concepts of networking and socket programming in C. By implementing both client and server components, the project highlights how data can be transmitted over a network in real time. Although UDP is chosen for its simplicity and speed, it is worth noting that it does not guarantee delivery or order of packets, which can be considered for future improvements. Overall, this project serves as a solid foundation for understanding network communications and paves the way for more advanced applications involving data transfer in different protocols.

REFERENCES

1. http://www.informit.com/store/tcp-ip illustrated-volume-1-the-protocols-9780201633610

2. https://www.rose-hulman.edu/~beecherm/UnixSockets/

3.https://www.oreilly.com/library/view/understanding-the-linux/9780596005658/

4. The Linux Programming Interface - https://man7.org/tlpi/

5.https://www.pearson.com/store/p/cprogramminglanguage/P100000920