|  |
| --- |
|  |
| Reliable UDP File Transfer |
| Project Report |

|  |
| --- |
| Group Members:  Muhammad Rouhan Faisal (370166)  Muhammad Abdullah (370146) |

# UDP File Transfer Application

## Introduction:

The UDP File Transfer Application is designed to facilitate the reliable transfer of files between a client and a server using the User Datagram Protocol (UDP). This document outlines the behavior, design, and development details of both the sender (client) and receiver (server) components.

## Sender (udp\_client.c):

## Overview:

The sender component is responsible for initiating the file transfer process. It establishes a connection with the server, divides the file into packets, and utilizes a sliding window protocol for efficient data transfer.

## Behavior:

**Connection Setup:**

* The client establishes a connection with the server using the socket and connect functions.
* Server information is obtained using getaddrinfo.

**File Transfer:**

* The client reads the file in chunks and sends packets to the server using sendto.
* It waits for acknowledgments for the sent packets using a separate thread (handleAcks).

**Sliding Window Protocol:**

* A sliding window approach is implemented to control the flow of data.
* Unacknowledged packets are resent if needed.

**Multithreading:**

* A thread is created to handle acknowledgments concurrently with the main sending process.

### Design:

**Packet Structure:**

* Each packet contains a sequence number, packet size, and data buffer.
* If the packet indicates the end of the file, the packet size is set to -1.

**Sliding Window:**

* A window of a fixed size (WINDOW\_SIZE) is used to control the number of unacknowledged packets.
* The window slides as acknowledgments are received.

**Thread for Acknowledgments:**

* A separate thread is created (handleAcks) to handle acknowledgments.
* It updates an acknowledgment array based on received acknowledgments.

**Timeout and Retransmission:**

* A timeout of 300 milliseconds is introduced using nanosleep.
* Unacknowledged packets are resent if the timeout occurs.

## Receiver (udp\_server.c):

### Overview:

The receiver component listens for incoming connections, receives files from clients, and acknowledges received packets. It also uses a sliding window protocol for efficient file reception.

### Behavior:

**Connection Setup:**

* The server establishes a socket, binds to an address, and enters a loop to continuously receive files from clients.

**File Reception:**

* The server receives the file size from the client.
* It waits for packets in chunks using recvfrom and saves them to the local file.

**Sliding Window Protocol:**

* Similar to the sender, the server uses a sliding window approach to manage incoming packets.
* It sends acknowledgments for received packets using a separate thread (handleAcks).

**Multithreading:**

* A thread is created to send acknowledgments concurrently with the main reception process.

### Design:

**Packet Structure:**

* Same as the sender, each packet contains a sequence number, packet size, and data buffer.
* The end of the file is indicated by a packet size of -1.

**Sliding Window:**

* A window of a fixed size (WINDOW\_SIZE) is used for managing incoming packets.
* The window slides as packets are received.

**Thread for Acknowledgments:**

* A separate thread is created (handleAcks) to send acknowledgments to the client.
* Acknowledgments are sent for a predefined number of packets.

**Timeout and Retransmission of Acknowledgments:**

* A timeout is not used for acknowledgments in the receiver.
* If the sender does not receive an acknowledgment, it resends the corresponding packet.

### Conclusion:

The UDP File Transfer Application demonstrates a reliable file transfer mechanism using UDP. The sliding window protocol and multithreading enhance efficiency and robustness. Developers can use this design as a foundation for further improvements and optimizations.

## Code for client side:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <fcntl.h>

#include <pthread.h>

#include <sys/stat.h>

#include <sys/sendfile.h>

#include <time.h>

#define SERVER\_PORT "4950"

#define BUFFER\_SIZE 500

#define WINDOW\_SIZE 5

*struct* packet {

*int* sequence\_no;

*int* packet\_size;

*char* data[BUFFER\_SIZE];

};

*struct* ThreadArgs {

*struct* packet \*packets;

*int* \*acks;

*int* \*no\_of\_acks;

};

*int* socket\_fd;

*struct* sockaddr\_storage server\_addr;

*socklen\_t* server\_addr\_len = sizeof(*struct* sockaddr\_storage);

*int* window\_start = 0;

*// Function to handle resending packets*

*void* resendPackets(*struct* packet *packets[]*, *int* *acks[]*, *int* *no\_of\_packets*) {

    for (*int* i = 0; i < *no\_of\_packets*; i++) {

        if (!*acks*[i]) {

            printf("Resending packet: %d\n", *packets*[i].sequence\_no);

            if (sendto(socket\_fd, &*packets*[i], sizeof(*struct* packet), 0, (*struct* sockaddr \*)&server\_addr, server\_addr\_len) < 0) {

                perror("UDP Client: sendto");

                exit(1);

            }

        }

    }

}

*// Function to handle the reception of acknowledgments*

*void* \*handleAcks(*void* \**args*) {

*struct* ThreadArgs \*threadArgs = (*struct* ThreadArgs \*)*args*;

*int* temp\_ack;

    for (*int* i = 0; i < WINDOW\_SIZE; i++) {

        if (recvfrom(socket\_fd, &temp\_ack, sizeof(*int*), 0, (*struct* sockaddr \*)&server\_addr, &server\_addr\_len) < 0) {

            perror("UDP Client: recvfrom");

            exit(1);

        }

        if (!threadArgs->acks[temp\_ack]) {

            printf("Ack Received: %d\n", temp\_ack);

            threadArgs->acks[temp\_ack] = 1;

            (\*threadArgs->no\_of\_acks)++;

        }

    }

    pthread\_exit(NULL);

}

*// Function to send a file using UDP*

*void* sendFile(*const* *char* \**hostname*, *const* *char* \**filename*) {

    FILE \*file = fopen(*filename*, "rb");

    if (!file) {

        perror("Error opening the file");

        exit(1);

    }

    fseek(file, 0, SEEK\_END);

*off\_t* file\_size = ftell(file);

    fseek(file, 0, SEEK\_SET);

    printf("Size of Video File: %ld bytes\n", file\_size);

    if (sendto(socket\_fd, &file\_size, sizeof(*off\_t*), 0, (*struct* sockaddr \*)&server\_addr, server\_addr\_len) < 0) {

        perror("UDP Client: sendto");

        exit(1);

    }

*struct* packet packets[WINDOW\_SIZE];

*int* acks[WINDOW\_SIZE] = {0};

*int* no\_of\_acks = 0;

*int* remaining\_data = file\_size;

    while (remaining\_data > 0) {

        for (*int* i = 0; i < WINDOW\_SIZE; i++) {

*int* data = fread(packets[i].data, 1, BUFFER\_SIZE, file);

            packets[i].sequence\_no = window\_start + i;

            packets[i].packet\_size = data;

            if (data == 0) {

                printf("End of file reached.\n");

                packets[i].packet\_size = -1;

                no\_of\_acks = i + 1;

                remaining\_data = 0;

                break;

            }

*// Print the packet being sent*

            printf("Sending packet %d: %d bytes\n", packets[i].sequence\_no, packets[i].packet\_size);

        }

*struct* ThreadArgs threadArgs = {packets, acks, &no\_of\_acks};

*pthread\_t* thread\_id;

        if (pthread\_create(&thread\_id, NULL, handleAcks, (*void* \*)&threadArgs) != 0) {

            perror("UDP Client: pthread\_create");

            exit(1);

        }

*struct* timespec time1, time2;

        time1.tv\_sec = 0;

        time1.tv\_nsec = 300000000L;

        nanosleep(&time1, &time2);

        pthread\_cancel(thread\_id);

        resendPackets(packets, acks, no\_of\_acks);

        window\_start += WINDOW\_SIZE;

        no\_of\_acks = 0;

    }

    fclose(file);

}

*int* main(*int* *argc*, *char* \**argv[]*) {

    if (*argc* != 3) {

        fprintf(stderr, "UDP Client: usage: %s hostname filename\n", *argv*[0]);

        exit(1);

    }

*const* *char* \*hostname = *argv*[1];

*const* *char* \*filename = *argv*[2];

*struct* addrinfo hints, \*serv\_info;

    memset(&hints, 0, sizeof hints);

    hints.ai\_family = AF\_UNSPEC;

    hints.ai\_socktype = SOCK\_DGRAM;

    if (getaddrinfo(hostname, SERVER\_PORT, &hints, &serv\_info) != 0) {

        fprintf(stderr, "UDP Client: getaddrinfo: Failed to get server address\n");

        exit(1);

    }

    for (*struct* addrinfo \*ptr = serv\_info; ptr != NULL; ptr = ptr->ai\_next) {

        if ((socket\_fd = socket(ptr->ai\_family, ptr->ai\_socktype, ptr->ai\_protocol)) == -1) {

            perror("UDP Client: socket");

            continue;

        }

        if (connect(socket\_fd, ptr->ai\_addr, ptr->ai\_addrlen) == -1) {

            close(socket\_fd);

            perror("UDP Client: connect");

            continue;

        }

        memcpy(&server\_addr, ptr->ai\_addr, ptr->ai\_addrlen);

        break;

    }

    if (serv\_info == NULL) {

        fprintf(stderr, "UDP Client: Failed to create socket\n");

        exit(2);

    }

    freeaddrinfo(serv\_info);

    sendFile(hostname, filename);

    printf("\nFile transfer completed successfully!\n");

    close(socket\_fd);

    return 0;

}

## Code for server side:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <fcntl.h>

#include <pthread.h>

#include <sys/stat.h>

#include <sys/sendfile.h>

#include <time.h>

#define SERVER\_PORT "4950"

#define BUFFER\_SIZE 500

#define WINDOW\_SIZE 5

*struct* packet {

*int* sequence\_no;

*int* packet\_size;

*char* data[BUFFER\_SIZE];

};

*int* socket\_fd;

*struct* sockaddr\_storage client\_addr;

*socklen\_t* client\_addr\_len = sizeof(*struct* sockaddr\_storage);

*// Function to handle the reception of acknowledgments*

*void* \*handleAcks(*void* \**args*) {

*int* \*acks = (*int* \*)*args*;

    for (*int* i = 0; i < WINDOW\_SIZE; i++) {

*int* temp\_ack = i;

        if (sendto(socket\_fd, &temp\_ack, sizeof(*int*), 0, (*struct* sockaddr \*)&client\_addr, client\_addr\_len) < 0) {

            perror("UDP Server: sendto");

            exit(1);

        }

        acks[temp\_ack] = 1;

        printf("Ack Sent: %d\n", temp\_ack);

    }

    pthread\_exit(NULL);

}

*// Function to receive a file using UDP*

*void* receiveFile() {

*off\_t* file\_size;

    if (recvfrom(socket\_fd, &file\_size, sizeof(*off\_t*), 0, (*struct* sockaddr \*)&client\_addr, &client\_addr\_len) < 0) {

        perror("UDP Server: recvfrom");

        exit(1);

    }

    printf("Receiving a file of size %ld bytes\n", file\_size);

    FILE \*file = fopen("received\_file.mp4", "wb");

    if (!file) {

        perror("Error opening the file");

        exit(1);

    }

*struct* packet packets[WINDOW\_SIZE];

*int* acks[WINDOW\_SIZE] = {0};

*int* no\_of\_acks = 0;

    while (1) {

        for (*int* i = 0; i < WINDOW\_SIZE; i++) {

            if (recvfrom(socket\_fd, &packets[i], sizeof(*struct* packet), 0, (*struct* sockaddr \*)&client\_addr, &client\_addr\_len) < 0) {

                perror("UDP Server: recvfrom");

                exit(1);

            }

            if (packets[i].packet\_size == -1) {

                printf("End of file reached.\n");

                break;

            }

*// Print the received packet information*

            printf("Received packet %d: %d bytes\n", packets[i].sequence\_no, packets[i].packet\_size);

            fwrite(packets[i].data, 1, packets[i].packet\_size, file);

        }

*pthread\_t* thread\_id;

        if (pthread\_create(&thread\_id, NULL, handleAcks, (*void* \*)acks) != 0) {

            perror("UDP Server: pthread\_create");

            exit(1);

        }

        pthread\_join(thread\_id, NULL);

        for (*int* i = 0; i < WINDOW\_SIZE; i++) {

            if (!acks[i]) {

                printf("Resending acknowledgment: %d\n", i);

                if (sendto(socket\_fd, &i, sizeof(*int*), 0, (*struct* sockaddr \*)&client\_addr, client\_addr\_len) < 0) {

                    perror("UDP Server: sendto");

                    exit(1);

                }

            }

        }

        memset(acks, 0, sizeof(acks));

        if (packets[WINDOW\_SIZE - 1].packet\_size == -1) {

            break; *// End of file*

        }

    }

    fclose(file);

}

*int* main() {

*struct* addrinfo hints, \*serv\_info;

    memset(&hints, 0, sizeof hints);

    hints.ai\_family = AF\_UNSPEC;

    hints.ai\_socktype = SOCK\_DGRAM;

    hints.ai\_flags = AI\_PASSIVE;

    if (getaddrinfo(NULL, SERVER\_PORT, &hints, &serv\_info) != 0) {

        fprintf(stderr, "UDP Server: getaddrinfo: Failed to get server address\n");

        exit(1);

    }

    for (*struct* addrinfo \*ptr = serv\_info; ptr != NULL; ptr = ptr->ai\_next) {

        if ((socket\_fd = socket(ptr->ai\_family, ptr->ai\_socktype, ptr->ai\_protocol)) == -1) {

            perror("UDP Server: socket");

            continue;

        }

        if (bind(socket\_fd, ptr->ai\_addr, ptr->ai\_addrlen) == -1) {

            close(socket\_fd);

            perror("UDP Server: bind");

            continue;

        }

        break;

    }

    if (serv\_info == NULL) {

        fprintf(stderr, "UDP Server: Failed to create socket\n");

        exit(2);

    }

    freeaddrinfo(serv\_info);

    printf("UDP Server is waiting for connections...\n");

    while (1) {

        receiveFile();

    }

    close(socket\_fd);

    return 0;

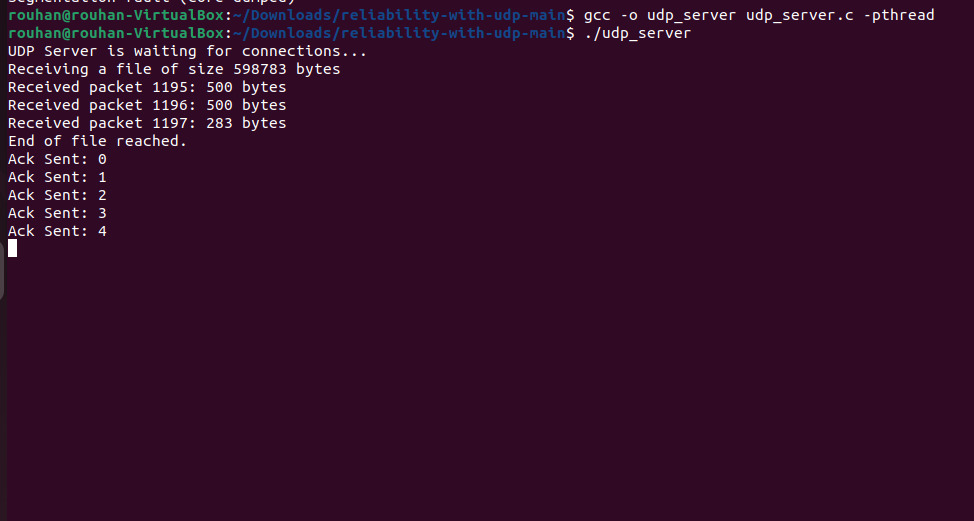
}

## Outputs

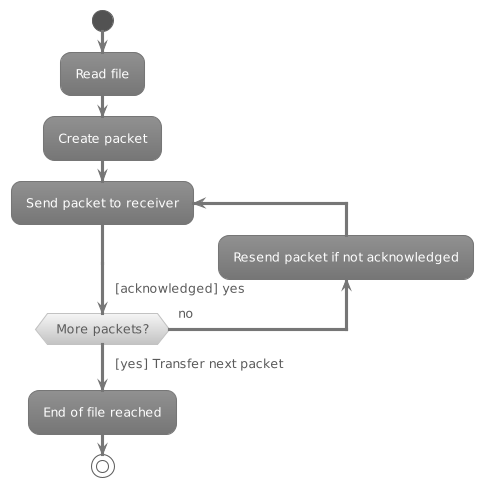
### Client Side:

### A screenshot of a computer Description automatically generated

### Server Side



## Flowchart



## Code Architecture

## Sequence Diagram

