Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 6

на тему «Элементы сетевого программирования»

Выполнил             Е. А. Киселёва

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 6](#_Toc157722975)

[Выводы 7](#_Toc157722976)

[Список использованных источников 9](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 10](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является построение системы обмена файлами клиентами через сеть с возможностью выбора и отправки файлов.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сеть – это типичная сложная система, которая состоит из подсистем, которые в свою очередь также являются сложными системами.

Существует два основных уровня компонентов сети:

– базовая сеть передачи данных;

– оконечные узлы.

Существует два основных способа передачи данных:

1 С установлением соединения: обеспечение целостности и упорядоченности потока передаваемых данных. Данные доставляются строго в том порядке, в котором были отправлены. Прерывание потока своевременно распознается. Это достигается за счет нумерации порций данных и организации встречного потока подтверждений о получении (квитанций). Образуется виртуальный канал передачи данных, который близок по свойствам к файлу или потоку ввода-вывода.

2 Без установления соединения: Данные передаются в виде датаграмм. Каждая датаграмма доставляется к получателю по произвольному маршруту и независимо от других датаграмм. Подтверждение о получении не предусматривается, поэтому не гарантируется ни порядок следования датаграмм, ни единственность доставленного экземпляра, ни сам факт доставки. Контролируются только искажения каждой отдельной датаграммы.

Открытая система – это система, построенная на основе открытых спецификаций, то есть таких, которые доступны для общего пользования и изменения. В контексте сетей, открытость есть доступность средств взаимодействия компонентов сети для общего использования и совместимость с другими системами.

Модель взаимодействия открытых систем OSI – это стандартизированная модель, разработанная Международной организаций по стандартизации ISO в 1983 году для описания взаимодействия различных устройств в сети.

Модель OSI разделяет коммуникационные функции в сети на уровни, начиная от физического соединения и заканчивая прикладными приложениями. Каждый уровень выполняет определенные функции, а взаимодействие между уровнями осуществляется через стандартизированные протоколы.

Сети, как правило, организованы иерархически, где различные уровни предоставляют разные уровни абстракции и функциональности. Это помогает в масштабировании и обслуживании сети.

Унификация в контексте сетей означает использование стандартных протоколов и спецификаций для обеспечения совместимости и взаимодействия между различными системами и компонентами сети.

Протокол – набор правил и процедур взаимодействия между одноименными уровнями различных систем, которые обеспечивают корректную связь участников взаимодействия в сети.

Интерфейс – набор правил и средств их реализации для взаимодействия между соседними уровнями одной системы, которые обеспечивают возможность модульного построения системы.

Стек протоколов в сети – набор протоколов, обслуживающих различные уровни взаимодействия. Протоколы в стеке проектируются с расчетом на совместную согласованную работу, но остаются достаточно независимыми для возможности замены на альтернативные с сохранением интерфейсов.

Сокет – программный объект, который обычно предоставляется операционной системой для создания сетевых соединений. Сокет скрывает детали реализации доступа к системе, позволяя приложениям взаимодействовать через сеть.[1]

Реализация сокетов на операционной системе Windows и Unix отличаются. На Windows сокеты реализуются при помощи подсистемы WinSock и соответствующих библиотек. На Unix же сокеты обычно реализованы в ядре и для их использования необходимы иные библиотеки.

Для выполнения данной лабораторной работы были использованы следующие сведения и концепции:

1 Создание сокета: в серверной части сначала создается сокет с помощью socket().

2 Привязка к адресу и порту: сервер привязывает свой сокет к определенному IP-адресу и порту с помощью bind(). Это позволяет серверу слушать входящие соединения на указанном порту.

3 Прослушивание соединений: функция listen() используется для настройки серверного сокета на прослушивание входящих соединений. Это позволяет серверу принимать соединения от клиентов.

4 Прием клиентов: в бесконечном цикле сервер ожидает входящие соединения с помощью accept(). Когда клиент подключается, создается новый поток (или потоки) для обработки взаимодействия с этим клиентом.

5 Обработка клиентов: в функции HandleClient происходит обработка команд от клиентов. Клиенты могут загружать файлы на сервер с командой "UPLOAD" и скачивать файлы с сервера с командой "DOWNLOAD". Сам файл передается в бинарном режиме, а данные читаются и записываются в буфер.

6 Управление клиентами: информация о клиентах хранится в векторе clients. Когда клиент отключается или происходит ошибка в соединении, его информация удаляется из вектора.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе выполнения лабораторной работы была построена система передачи файлов клиентами через сеть с возможностью выбора и отправки файлов. Проект поделен на две части:

– серверная часть;

– клиентская часть.

В систему входят три проекта. Проект Server отображает серверную часть, а проекты Client1 и Client2 – клиентскую часть.

Результат успешного запуска сервера и подключения к нему клиента представлен на рисунке 3.1.

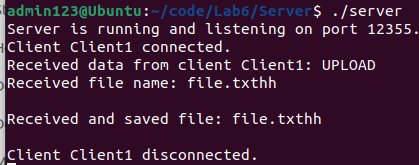


Рисунок 3.1 – Результат работы сервера при запуске и подключения к нему клиентов

Отправка файлов от клиента осуществляется при помощи использования консольного меню. Результат отправки файлов от клиента к серверу представлен на рисунке 3.2.

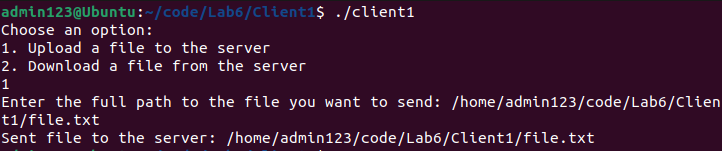


Рисунок 3.2 – Результат подключения клиентов к серверу

Для отправки файлов от сервера, клиенту достаточно ввести имя файла, который будет расположен на сервере. Результат получения файлов от сервера представлен на рисунке 3.3.

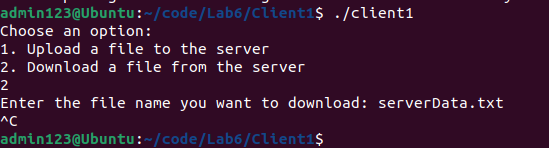


Рисунок 3.3 – Результат получения файлов от сервера

Также отправка файлов клиенту фиксируется на стороне сервера. Фиксация результатов отправки файлов от сервера представлен на рисунке 3.4.

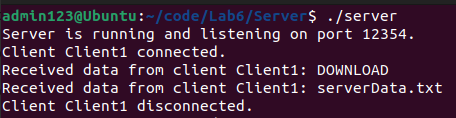


Рисунок 3.4 – Отправка файлов от сервера

Таким образом была разработана система обмена файлами между клиентами через сеть.

# **ВЫВОДЫ**

В ходе выполнения данной лабораторной работы была построена система передачи файлов клиентами по сети с возможность выбора и отправки файлов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Сокеты [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/otus/articles/539550/. – Дата доступа: 23.03.2024.

[2] В чем отличие сокетов Windows от Linux [Электронный ресурс]. – Режим доступа: https://forum.ixbt.com/topic.cgi?id=26:42885. – Дата доступа: 24.03.2024.

[3] TCP [Электронный ресурс]. – Режим доступа: https://stormwall.pro/knowledge-base/termin/tcp-handshake. – Дата доступа: 23.03.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <pthread.h>

#define PORT 12354

#define MAX\_CLIENTS 10

#define BUFFER\_SIZE 1024

struct ClientInfo {

char name[BUFFER\_SIZE];

int socket;

};

struct ClientInfo clients[MAX\_CLIENTS];

int client\_count = 0;

void HandleClient(struct ClientInfo \*client) {

char buffer[BUFFER\_SIZE];

int bytesRead;

while (1) {

bytesRead = recv(client->socket, buffer, BUFFER\_SIZE, 0);

if (bytesRead <= 0) {

printf("Client %s disconnected.\n", client->name);

close(client->socket);

for (int i = 0; i < client\_count; ++i) {

if (clients[i].socket == client->socket) {

for (int j = i; j < client\_count - 1; ++j) {

clients[j] = clients[j + 1];

}

client\_count--;

break;

}

}

return;

}

buffer[bytesRead] = '\0';

printf("Received data from client %s: %s\n", client->name, buffer);

if (strcmp(buffer, "UPLOAD") == 0) {

bytesRead = recv(client->socket, buffer, BUFFER\_SIZE, 0);

if (bytesRead <= 0) {

perror("Error receiving file name");

return;

}

buffer[bytesRead] = '\0';

printf("Received file name: %s\n", buffer);

FILE\* fileStream = fopen(buffer, "wb");

if (!fileStream) {

perror("Error opening file for writing");

return;

}

while ((bytesRead = recv(client->socket, buffer, BUFFER\_SIZE, 0)) > 0) {

fwrite(buffer, 1, bytesRead, fileStream);

}

fclose(fileStream);

printf("Received and saved file: %s\n", buffer);

}

}

}

void \*HandleClientThread(void \*arg) {

struct ClientInfo \*pClient = (struct ClientInfo \*)arg;

HandleClient(pClient);

return NULL;

}

int main() {

int serverSocket, clientSocket;

struct sockaddr\_in serverAddr, clientAddr;

socklen\_t clientAddrSize = sizeof(clientAddr);

pthread\_t tid;

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Error creating socket");

exit(EXIT\_FAILURE);

}

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(PORT);

if (bind(serverSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) == -1) {

perror("Error binding socket");

exit(EXIT\_FAILURE);

}

if (listen(serverSocket, MAX\_CLIENTS) == -1) {

perror("Error listening to socket");

exit(EXIT\_FAILURE);

}

printf("Server is running and listening on port %d.\n", PORT);

while (1) {

if ((clientSocket = accept(serverSocket, (struct sockaddr \*)&clientAddr, &clientAddrSize)) == -1) {

perror("Error accepting connection");

exit(EXIT\_FAILURE);

}

char nameBuffer[BUFFER\_SIZE];

int nameBytes = recv(clientSocket, nameBuffer, BUFFER\_SIZE, 0);

if (nameBytes > 0) {

nameBuffer[nameBytes] = '\0';

}

struct ClientInfo newClient;

strcpy(newClient.name, nameBuffer);

newClient.socket = clientSocket;

clients[client\_count++] = newClient;

if (pthread\_create(&tid, NULL, HandleClientThread, (void \*)&newClient) != 0) {

perror("Error creating thread");

exit(EXIT\_FAILURE);

}

printf("Client %s connected.\n", newClient.name);

}

close(serverSocket);

return 0;

}

Листинг 2 – Программный код client1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define BUFFER\_SIZE 1024

void SendCommand(int clientSocket, const char\* command) {

if (send(clientSocket, command, strlen(command), 0) < 0) {

perror("Error sending the command");

exit(EXIT\_FAILURE);

}

}

void SendFileName(int clientSocket, const char\* fileName) {

if (send(clientSocket, fileName, strlen(fileName), 0) < 0) {

perror("Error sending the file name");

exit(EXIT\_FAILURE);

}

}

void SendFile(int clientSocket, const char\* filePath) {

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead;

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

if (!file) {

perror("Error opening file for reading");

return;

}

, '/') + 1;

SendFileName(clientSocket, fileName);

while ((bytesRead = fread(buffer, 1, BUFFER\_SIZE, file)) > 0) {

if (send(clientSocket, buffer, bytesRead, 0) < 0) {

perror("Error sending file data");

fclose(file);

return;

}

}

fclose(file);

printf("Sent file to the server: %s\n", filePath);

}

void ReceiveFile(int clientSocket) {

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead;

char downloadCommand[] = "DOWNLOAD";

if (send(clientSocket, downloadCommand, sizeof(downloadCommand), 0) < 0) {

fprintf(stderr, "Error sending the download command.\n");

return;

}

printf("Enter the file name you want to download: ");

char fileNameToDownload[BUFFER\_SIZE];

scanf("%s", fileNameToDownload);

if (send(clientSocket, fileNameToDownload, strlen(fileNameToDownload), 0) < 0) {

fprintf(stderr, "Error sending the file name to download.\n");

return;

}

FILE \*receivedFile = fopen(fileNameToDownload, "wb");

if (!receivedFile) {

fprintf(stderr, "Error opening file for writing: %s\n", fileNameToDownload);

return;

} else {

while ((bytesRead = recv(clientSocket, buffer, BUFFER\_SIZE, 0)) > 0) {

fwrite(buffer, 1, bytesRead, receivedFile);

}

fclose(receivedFile);

printf("Received and saved file: %s\n", fileNameToDownload);

}

}

int main() {

int clientSocket;

struct sockaddr\_in serverAddr;

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == -1) {

perror("Error creating socket");

exit(EXIT\_FAILURE);

}

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serverAddr.sin\_port = htons(12354);

if (connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == -1) {

perror("Error connecting to the server");

close(clientSocket);

exit(EXIT\_FAILURE);

}

char name[] = "Client1";

send(clientSocket, name, strlen(name), 0);

printf("Choose an option:\n1. Upload a file to the server\n2. Download a file from the server\n");

int option;

scanf("%d", &option);

if (option == 1) {

SendCommand(clientSocket, "UPLOAD");

char filePath[BUFFER\_SIZE];

printf("Enter the full path to the file you want to send: ");

scanf("%s", filePath);

getchar();

SendFile(clientSocket, filePath);

} else if (option == 2) {

ReceiveFile(clientSocket);

} else {

fprintf(stderr, "Invalid option.\n");

}

close(clientSocket);

return 0;

}

Листинг 3 – Программный код client2.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define BUFFER\_SIZE 1024

void SendFile(int clientSocket, const char\* filePath) {

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead; // Байты, считанные из файла

char \*fileName = strrchr(filePath, '/');

if (!fileName) {

fileName = strrchr(filePath, '\\');

if (!fileName) {

fileName = filePath;

} else {

fileName++;

}

} else {

fileName++;

}

char uploadCommand[] = "UPLOAD";

if (send(clientSocket, uploadCommand, sizeof(uploadCommand), 0) < 0) {

fprintf(stderr, "Error sending the upload command.\n");

return;

}

if (send(clientSocket, fileName, strlen(fileName), 0) < 0) {

fprintf(stderr, "Error sending the file name.\n");

return;

}

FILE \*fileStream = fopen(filePath, "rb");

if (!fileStream) {

fprintf(stderr, "Error opening file for reading: %s\n", filePath);

return;

}

while ((bytesRead = fread(buffer, 1, BUFFER\_SIZE, fileStream)) > 0) {

if (send(clientSocket, buffer, bytesRead, 0) < 0) {

fprintf(stderr, "Error sending file data.\n");

break;

}

}

fclose(fileStream);

printf("Sent file to the server: %s\n", filePath);

}

void ReceiveFile(int clientSocket) {

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead;

char downloadCommand[] = "DOWNLOAD";

if (send(clientSocket, downloadCommand, sizeof(downloadCommand), 0) < 0) {

fprintf(stderr, "Error sending the download command.\n");

return;

}

printf("Enter the file name you want to download: ");

char fileNameToDownload[BUFFER\_SIZE];

scanf("%s", fileNameToDownload);

if (send(clientSocket, fileNameToDownload, strlen(fileNameToDownload), 0) < 0) {

fprintf(stderr, "Error sending the file name to download.\n");

return;

}

FILE \*receivedFile = fopen(fileNameToDownload, "wb");

if (!receivedFile) {

fprintf(stderr, "Error opening file for writing: %s\n", fileNameToDownload);

return;

} else {

while ((bytesRead = recv(clientSocket, buffer, BUFFER\_SIZE, 0)) > 0) {

fwrite(buffer, 1, bytesRead, receivedFile);

}

fclose(receivedFile);

printf("Received and saved file: %s\n", fileNameToDownload);

}

}

int main() {

int clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket < 0) {

fprintf(stderr, "Error creating a socket.\n");

return 1;

}

struct sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serverAddr.sin\_port = htons(12345);

if (connect(clientSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) < 0) {

fprintf(stderr, "Error connecting to the server.\n");

close(clientSocket);

return 1;

}

char name[] = "Client2";

send(clientSocket, name, sizeof(name), 0);

printf("Choose an option:\n1. Upload a file to the server\n2. Download a file from the server\n");

int option;

scanf("%d", &option);

if (option == 1) {

printf("Enter the full path to the file you want to send: ");

char filePath[BUFFER\_SIZE];

scanf("%s", filePath);

SendFile(clientSocket, filePath);

} else if (option == 2) {

ReceiveFile(clientSocket);

} else {

fprintf(stderr, "Invalid option.\n");

}

close(clientSocket);

return 0;

}