Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**Клиент-сервер. Создание Web-серверов.**

Лабораторная работа №12 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_ Баранов А. С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:**

В результате выполнения программы, объединить полученные в процессе выполнения предыдущих лабораторных работ знания в рамках одной комплексной программы.

**Задание:**

Написать программу-клиент и программу-сервер для выполнения следующей задачи: сервер запускается и ждёт запросов от клиентов; клиент подключается к серверу и может запрашивать файлы, находящиеся на сервере (в папке с программой-сервером), задавая полное имя файла; если файл найден, сервер передаёт файл клиенту, если не найден, то сервер возвращает сообщение о том, что файл не найден.

Сервер должен параллельно обслуживать несколько клиентов (для каждого свой процесс). Клиент может запрашивать файлы последовательно вводя имена. Выход из программы клиента можно осуществить по текстовой команде (например, «exit»). Сервер так же должен иметь возможность вводить команды (например, для закрытия программы-сервера).

Протокол транспортного уровня – TCP.

При запуске программы-сервера через параметры командной строки передаётся порт, на котором будет работать сервер. Если данный порт занят, программа-сервер должна получить порт по усмотрению системы. При успешном запуске сервер должен выдать на экран номер порта, на котором он работает.

При запуске программы-клиента через параметры командной строки передаётся IP-адрес и порт (сокет) программы-сервера в формате <IP-address>:<port> (например, 192.168.0.100:4242).

**Содержание server.c**

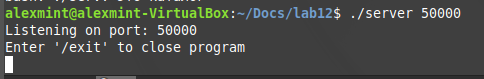
|  |
| --- |
| #include <stdio.h> |
| #include <string.h> |
| #include <stdlib.h> |
| #include <time.h> |
| #include <sys/types.h> |
| #include <sys/socket.h> |
| #include <sys/stat.h> |
| #include <fcntl.h> |
| #include <netinet/in.h> |
| #include <arpa/inet.h> |
| #include <unistd.h> |
| #include <iostream> |
|  |
| using namespace std; |
|  |
| long getFileSize(FILE\* file) |
| { |
| fseek(file, 0L, SEEK\_END); |
| long size = ftell(file); |
| rewind(file); |
| return size; |
| } |
|  |
|  |
| int main(int argc, char\* argv[]) |
| { |
| int port = atoi(argv[1]); |
| struct sockaddr\_in server\_addr; |
|  |
| bzero(&server\_addr, sizeof(server\_addr)); |
| server\_addr.sin\_family = PF\_INET; |
| server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); |
| server\_addr.sin\_port = htons(port); |
|  |
| int sock = socket(PF\_INET, SOCK\_STREAM, 0); |
| if (sock < 0) |
| { |
| perror("socket() error"); |
| exit(-1); |
| } |
| if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) |
| { |
| server\_addr.sin\_port = 0; |
| if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) |
| { |
| perror("bind() error"); |
| exit(-1); |
| } |
| } |
|  |
| socklen\_t server\_len = sizeof(server\_addr); |
| listen(sock, 100); |
| getsockname(sock, (struct sockaddr \*)&server\_addr, &server\_len); |
| cout << "Listening on port: " << ntohs(server\_addr.sin\_port) << endl; |
|  |
| if (fork() == 0) |
| { |
| while (1) |
| { |
| struct sockaddr\_in client\_addr; |
| socklen\_t client\_len = sizeof(client\_addr); |
| int newSock = accept(sock, (struct sockaddr \*)&client\_addr, &client\_len); |
|  |
| if (fork() == 0) |
| { |
| continue; |
| } |
| while (1) |
| { |
| char input[128]; |
| int n = read(newSock, input, sizeof(input)); |
| if (n == 0) |
| { |
| close(newSock); |
| exit(0); |
| } |
|  |
| FILE\* file = fopen(input, "r"); |
| if (!file) |
| { |
| long fileSize = -1 ; |
| write(newSock, &fileSize, sizeof(fileSize)); |
| } |
| else |
| { |
|  |
| long fileSize = getFileSize(file); |
| write(newSock, &fileSize, sizeof(fileSize)); |
| char message[fileSize]; |
| fread(message, sizeof(char), fileSize, file); |
| fclose(file); |
| write(newSock, message, sizeof(message)); |
| } |
| } |
| } |
| } |
| else |
| { |
| cout << "Enter '/exit' to close program" << endl; |
| char message[100]; |
| while (1) |
| { |
| cin >> message; |
|  |
| if (!strcmp(message, "/exit")) |
| { |
| close(sock); |
| exit(0); |
| } |
| else |
| { |
| cout << "You entered a nonexistent command" << endl; |
| } |
| } |
| } |
| } |

**Содержание client.c**

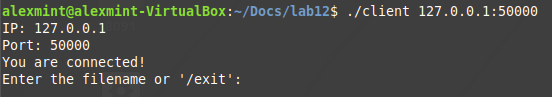
|  |
| --- |
| #include <stdio.h> |
| #include <string.h> |
| #include <stdlib.h> |
| #include <time.h> |
| #include <sys/types.h> |
| #include <sys/socket.h> |
| #include <sys/stat.h> |
| #include <fcntl.h> |
| #include <netinet/in.h> |
| #include <arpa/inet.h> |
| #include <unistd.h> |
| #include <iostream> |
|  |
| using namespace std; |
|  |
|  |
| void recvFile(int sock, char\* filename, long fileSize) |
| { |
|  |
| } |
|  |
| int ipget(char\* arg, char\* ip) |
| { |
| int iplen; |
| int i = 0; |
| while(arg) |
| { |
| if(arg[i] == ':') |
| { |
| iplen = i; |
| break; |
| }i++; |
| } i = 0; |
| for (char\* l = ip; l < ip + iplen ; l++) |
| { |
| \*l = \*(arg + i); |
| i++; |
| }\*(ip + i) = '\0'; |
|  |
| return iplen; |
| } |
|  |
| int main(int argc, char\* argv[]) |
| { |
| if(argc != 2 ) |
| {cout << "Error: program needs socket." << endl;} |
| int ipl; char ip[15]; |
| ipl = ipget(argv[1], ip); |
| char\* prt = new char[strlen(argv[1]) - ipl + 1]; |
| prt = argv[1] + ipl+1; |
| cout << "IP: " << ip << endl; |
| cout << "Port: " << prt << endl; |
|  |
| struct sockaddr\_in server\_addr; |
| int sockfd = socket(PF\_INET, SOCK\_STREAM, 0); |
| if (sockfd < 0) |
| { |
| perror("socket() error"); |
| exit(-1); |
| } |
|  |
| bzero(&server\_addr, sizeof(server\_addr)); |
| server\_addr.sin\_family = PF\_INET; |
| server\_addr.sin\_port = htons(atoi(prt)); |
| inet\_aton(ip, &server\_addr.sin\_addr); |
|  |
| if(connect(sockfd, (struct sockaddr\*) &server\_addr, sizeof(server\_addr)) < 0) |
| { |
| perror("connect() error"); |
| close(sockfd); |
| exit(-1); |
| } |
|  |
| cout << "You are connected!" << endl; |
| cout << "Enter the filename or '/exit': " << endl; |
|  |
| char message[128]; |
| while (1) |
| { |
| cin >> message; |
|  |
| if (!strcmp(message,"/exit")) |
| { |
| cout << "Good bye!" << endl; |
| close(sockfd); |
| exit(0); |
| } |
| else |
| { |
| long fileSize; |
| write(sockfd, message, strlen(message) + 1); |
| int n = read(sockfd, &fileSize, sizeof(long)); |
| if (n == 0) |
| { |
| close(sockfd); |
| perror("Connection lost"); |
| exit(0); |
| } |
| if (fileSize < 0) |
| { |
| cout << "The file is busy with another program or does not exist" << endl; |
| } |
| else |
| { |
| recvFile(sockfd, message, fileSize); |
|  |
|  |
| char buffer[fileSize]; |
| bzero(buffer, fileSize); |
| FILE\* file = fopen(message, "w"); |
| cout << endl; |
| int n = 0; |
| while (n < fileSize) |
| { |
| n += read(sockfd, buffer, sizeof(buffer)); |
| fprintf(file, "%s", buffer); |
| } |
| for(int i = 0; i < fileSize; i++) |
| { |
| cout << buffer[i]; |
| } |
| truncate(message, fileSize); |
| cout << endl; |
| fclose(file); |
| cout << "File received successfully as " << message << endl; |
| } |
| } |
| } |
| } |

**Результат выполнения программы:**

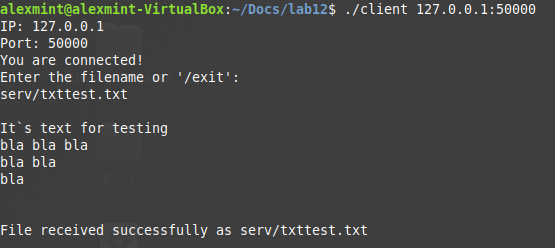
Запуск сервера:



Работа клиента:



Запрос и получение файла:



**Вывод:**

В результате выполнения данной лабораторной работы, я на практике закрепил знания о различных аспектах UNIX и научился работать с буфером.