|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №3**

Студент Журавлев Николай Вадимович

Группа РК6-62б

Тип задания Лабораторная работа

Тема лабораторной работы Сетевое программирование

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Н.В. Журавлев**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_В.Г. Федорук\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2022 г.*

Оглавление

[Текст задания 3](#_Toc104199823)

[Описание используемого прикладного протокола сетевого взаимодействия 3](#_Toc104199824)

[Описание структуры программы 4](#_Toc104199825)

[Описание основных используемых структур данных 5](#_Toc104199826)

[Блок-схема программы 5](#_Toc104199827)

[Примеры результатов работы программы 8](#_Toc104199828)

[Текст программы 8](#_Toc104199829)

# Текст задания

Разработать клиент-серверное приложение копирования файла (или поддерева файловой системы) с узла-сервера на узел-клиент в указанный каталог (аналог стандартной UNIX-команды rcp). Команда, выполняемая на стороне клиента, имеет следующий вид: remcp host@path.to.src.file path.to.dst.dir.

Замечание. Для передачи поддерева файловой системы (или одного файла) рекомендуется на стороне клиента использовать вызов (через fork) команды tar с перенаправлением ее вывода непосредственно в сокет. На стороне сервера также рекомендуется использовать tar с перенаправлением ее ввода непосредственно из сокет.

# Описание используемого прикладного протокола сетевого взаимодействия

Протокол TCP – один из основных транспортных протоколов. Данный протокол работает на основе потока данных с установкой соединения и гарантированной передаче данных – при потере данных осуществляется повторный запрос, а также устраняется дублирование при получении двух копий одного пакета.

Установку соединения обеспечивает механизм сокетов, которые описываются в библиотеке sys/socket.h. Для создания TCP сокета используется функция int socket (AF\_INET, SOCK\_STREAM, 0). Детали установки и проверки соединения зашиты в функцию, которая возвращает дескриптор соокета, если соединение успешно.

После создания, сокет необходимо связать с коммуникационной средой. Делается это с помощью функции int bind (s, addr, addrlen), где s – дескриптор сокета, addr – структура адреса сокета, addrlen – размер addr.

Клиент-серверная архитектура требует, соответственно, наличие клиента и сервера. После создания и привязки сокета серверу необходимо прослушать входящие соединения. Делается это с помощью функции int listen (s, n), где s – дескриптор сокета, n – максимальная длина очереди клиентов. Когда на сервер стучится клиент, серверу необходимо принять клиента и установить с ним связь. Для этого используется функция int accept (s, addr, p\_addrlen), где s – дескриптор сокета сервера, addr – структура, куда будет записан адрес клиента, а p\_addrlen – размер этой структуры. Данная функция вернёт сокет подключившегося клиента.

В свою очередь, клиенту, после создания и привязки сокета необходимо обратиться с запросом на подключение к серверу. Для этого клиент вызывает функцию int connect (s,addr,addrlen), где s – дескриптор сокета клиента, addr – структура, куда будет записан адрес сервера, а p\_addrlen – размер этой структуры. Данная функция вернёт сокет подключившегося сервера.

В процессе взаимодействия клиент и сервер посылают друг другу данные. Функция int send (s,buf,len,flags) позволяет отправить данные по сокету s, которые содержатся в буфере buf размером len, а аргумент flags модифицирует выполнение функции send. Функция int recv (s,buf,len,flags) позволяет получить данные по сокету s, записав их в буфер buf размером len, а аргумент flags модифицирует.

# Описание структуры программы

Программа-клиент отправляет название директории, которую необходимо копировать, на сервер. Далее она создает и открывает пустой файл-архив (архив создается в папке назначения), в который записывает данные, которые присылает сервер. Затем программа-клиент распаковывает полученный архив и удаляет его с помощью команды tar.

Программа-сервер получает от клиента название директории, которую необходимо передать. Программа архивирует файл и помещает архив в директорию (путь до этой директории задается программистом в тексте программы) с помощью команды tar. Затем сервер отправляет на клиент данные созданного архива. После того, как весь архив отправлен, программа удаляет его и закрывает соединение с клиентом.

# Описание основных используемых структур данных

С помощью define задаются порты клиента и сервера, размер буфера, размер команды и длина пути.

Для получения от клиента адреса копируемой папки на сервере используется системный вызов recv(s\_new, source, NAMES, 0), где s\_new дескриптор сокета, возвращенный системным вызовом s\_new = accept(s, &from\_sin, &from\_len),

На клиенте, для того, чтобы послать адрес папки используется вызов send(s, source, NAMES, 0), где s = socket(AF\_INET, SOCK\_STREAM, 0) дескриптор сокета.

Изображение взаимодействия клиента и сервера представлено на рис.1.

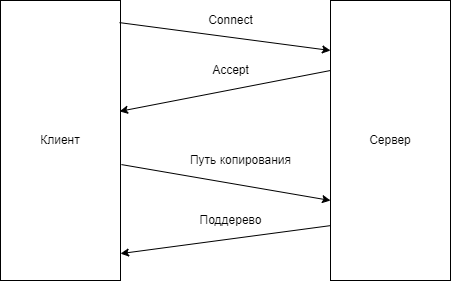


Рисунок . Схема взаимодействия клиента и сервера

# Блок-схема программы

Блок-схема реализованных клиента и сервера представлены на рисунке 2 и 3.



Рисунок . Блок-схема клиента



Рисунок . Блок-схема сервера

# Примеры результатов работы программы

Результатом работы разработанной программы с пустой папкой 2 при аргументах 127.0.0.1@/home/.../Desktop/1 /home/.../Desktop/2 представлены на рисунках 4, 5.

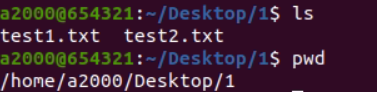


Рисунок . Начальное расположение файлов

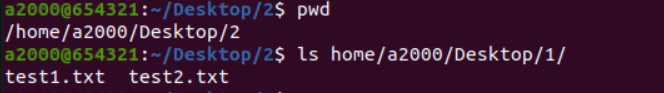


Рисунок . Результат работы программы

# Текст программы

Код клиента:

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#define SRV\_PORT 1234 // порт сервера

#define CLNT\_PORT 1235 // порт клиента

#define PATH\_SIZE 64 // размер пути

#define TAR\_C 256 // размер команды архиватора

#define TAR "tar -x <&" // утилита архиватор prev

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

int s;

char path[PATH\_SIZE]; // путь

char \*path\_p;

char tar[TAR\_C];

char host[TAR\_C]; // адрес сервера

char number[20];

struct hostent \*hp = NULL;

struct sockaddr\_in clnt\_sin, srv\_sin;

if (argc < 3) {

write(1, "I need more arguments\n", 23);

exit(1);

}

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

memset((char \*) &clnt\_sin, '\0', sizeof(clnt\_sin));

clnt\_sin.sin\_family = AF\_INET;

clnt\_sin.sin\_addr.s\_addr = INADDR\_ANY;

clnt\_sin.sin\_port = CLNT\_PORT;

bind(s, (struct sockaddr \*) &clnt\_sin, sizeof(clnt\_sin));

memcpy(host, argv[1], strlen(argv[1]));

\*strchr(host, '@') = '\0';

memset((char \*) &srv\_sin, '\0', sizeof(srv\_sin));

if ((hp = gethostbyname(host)) == NULL) { // получаем host

write(1, "Can't find ", 11);

write(1, host, strlen(host));

write(1, "\n", 1);

return 1;

}

srv\_sin.sin\_family = AF\_INET;

memcpy((char \*) &srv\_sin.sin\_addr, hp->h\_addr, hp->h\_length);

srv\_sin.sin\_port = SRV\_PORT;

if (connect(s, (struct sockaddr \*) &srv\_sin, sizeof(srv\_sin)) == -1) { // подключение

write(1, "Can't connect to ", 17);

write(1, host, strlen(host));

write(1, "\n", 1);

return 1;

}

path\_p = strstr(argv[1], "@");

path\_p++;

memcpy(path, path\_p, strlen(path\_p));

path[strlen(path\_p)] = '\0';

write(s, path, strlen(path));

if (chdir(argv[2]) < 0) {

return (-4);

}

sprintf(number,"%d", s);

memcpy(tar, TAR, strlen(TAR));

memcpy(tar + strlen(TAR), number, strlen(number));

tar[strlen(TAR) + strlen(number)] = '\0';

system(tar);

exit(0);

}

Код сервера:

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <signal.h>

#define SRV\_PORT 1234 // порт подключения

#define BUF\_SIZE 4096 // размер буфера

#define PATH\_SIZE 64 // размер пути

#define TAR\_C 128 // длина команды tar

#define TAR "tar -c -O " // команда архивации

#define REDIRECT " 1>&"

int s, s\_new = 0;

int from\_len;

char path[PATH\_SIZE];

char tar[TAR\_C];

void siginthandler() { // обработчик Ctrl+C, грамотное отключение сервера

write(1, "\nServer is shutting down\n", 25);

shutdown(s\_new, 0);

shutdown(s, 2);

if(s\_new) close(s\_new);

exit(0);

}

int main () {

int i;

char number[20];

struct sockaddr\_in sin;

signal(SIGINT, &siginthandler); // установка обработчика SIGINT

s = socket(AF\_INET, SOCK\_STREAM, 0); // установка сокета

memset((char \*)&sin, '\0', sizeof(sin)); // обнуление структуры

sin.sin\_family = AF\_INET;

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

sin.sin\_port = SRV\_PORT;

bind(s, (struct sockaddr \*)&sin, sizeof(sin));

listen(s, 3);

while(1) {

struct sockaddr\_in from\_sin;

int path\_l;

for(i = 0; i < PATH\_SIZE; i++) {

path[i] = '\0';

}

from\_len = sizeof(from\_sin);

s\_new = accept(s, (struct sockaddr \*) &from\_sin, &from\_len);

path\_l = read(s\_new, path, PATH\_SIZE);

if (path\_l == 0) {

continue;

}

chdir(path);

sprintf(number,"%d", s\_new);

memcpy(tar, TAR, strlen(TAR));

memcpy(tar + strlen(TAR), path, strlen(path));

memcpy(tar + strlen(TAR) + strlen(path), REDIRECT, strlen(REDIRECT));

memcpy(tar + strlen(TAR) + strlen(path) + strlen(REDIRECT), number, strlen(number));

tar[strlen(TAR) + strlen(path) + strlen(REDIRECT) + strlen(number)] = '\0';

system(tar);

write(1, path, strlen(path));

shutdown(s\_new, 2); // отключение подключения

close(s\_new);

s\_new = 0;

}

}