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

ОТЧЁТ

к лабораторной работе №6

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил студент гр.153502 Толстой Д. В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc158209942)

[2 Описание функций программы 4](#_Toc158209943)

[Список использованных источников 6](#_Toc158209944)

[Приложение А (обязательное) Листинг кода 7](#_Toc158209945)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Разработка и реализация командного интерпретатора («оболочки», *shell*) с «дистанционным управлением» (включая протокол прикладного уровня для взаимодействия с ним). Язык интерпретатора редуцированный – для целей демонстрации. О безопасности соединения не обязательно (т.е. делать *ssh* не нужно).

Агент (сервер): установка на заданный порт; прием запросов (команд для исполнения); исполнение поступивших команд в локальной системе; возврат результата (опционально, если предусмотрено для конкретной команды).

Менеджер (клиент): соединение с агентом; ввод команд; передача команд агенту; контроль результативности.

Набор команд может быть как собственный (синтезированный), так и совпадающий с «родными» командами в среде локальной системы. Во втором случае для их выполнения можно воспользоваться вызовом *system*().

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа создает сервер, который слушает входящие соединения на порту 8081 и выполняет команды, полученные от клиента, выводя результат выполнения команд обратно клиенту. Сервер инициализирует сокет, привязывает его к адресу и порту, затем начинает слушать входящие соединения. Когда клиент подключается, сервер читает команду, отправленную клиентом, и выполняет ее с помощью функции *system*(), перенаправляя вывод в файл. Затем сервер читает содержимое файла и отправляет его обратно клиенту. После отправки вывода сервер освобождает выделенную память и закрывает соединение с клиентом.

Клиентская программа подключается к серверу по адресу 127.0.0.1 и порту 8081, отправляет команды для выполнения на удаленной системе и получает результаты выполнения команд обратно от сервера. Клиент инициализирует сокет, устанавливает соединение с сервером, затем вводит команды для выполнения и отправляет их серверу. После отправки команды клиент читает и выводит результат выполнения команды, полученный от сервера. Клиент продолжает отправлять команды до тех пор, пока не введет команду *exit* для выхода из программы.

Пример работы сервера представлен на рисунке 1.

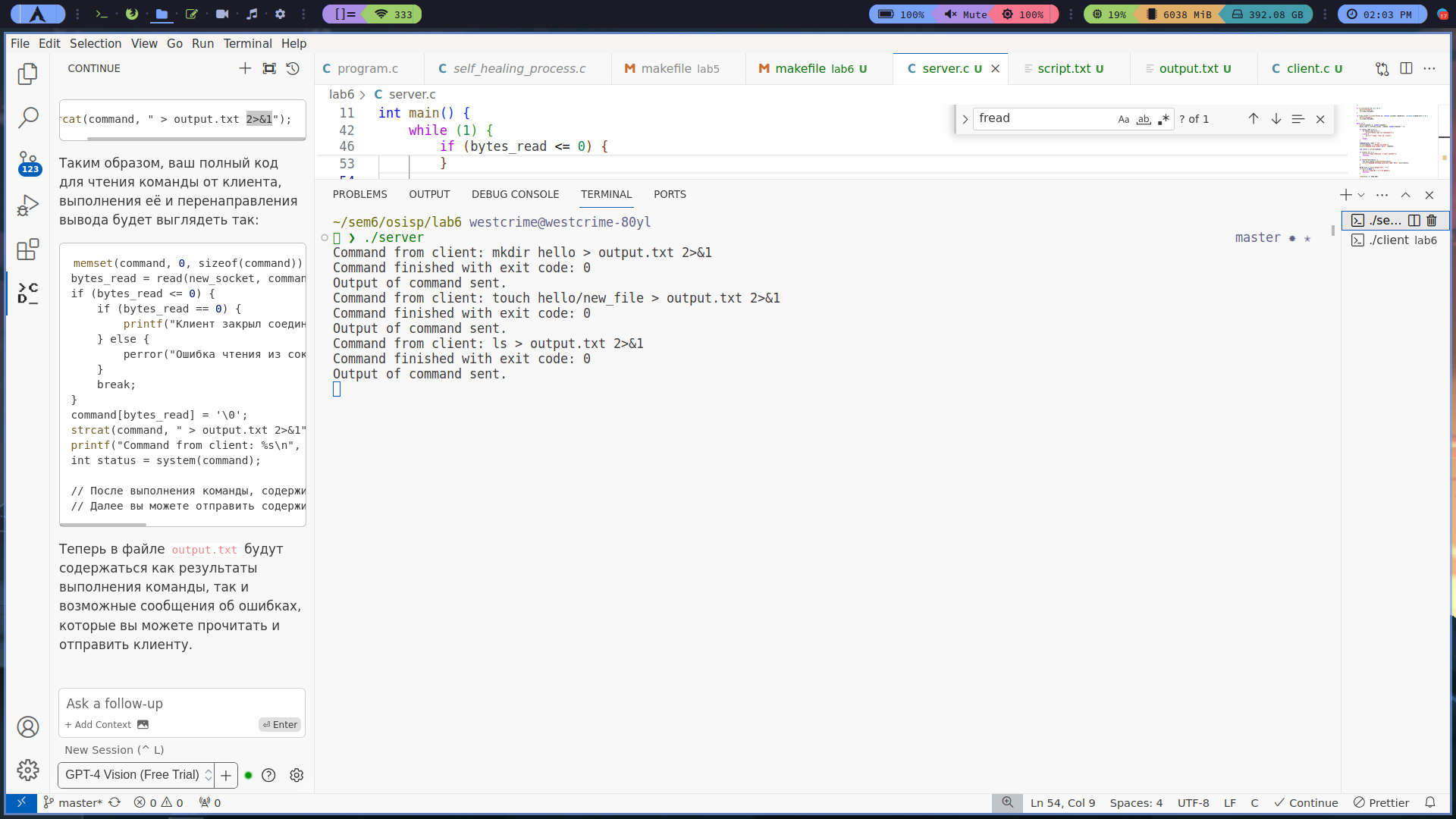


Рисунок 1 – Пример работы сервера

Пример работы клиента представлен на рисунке 2.

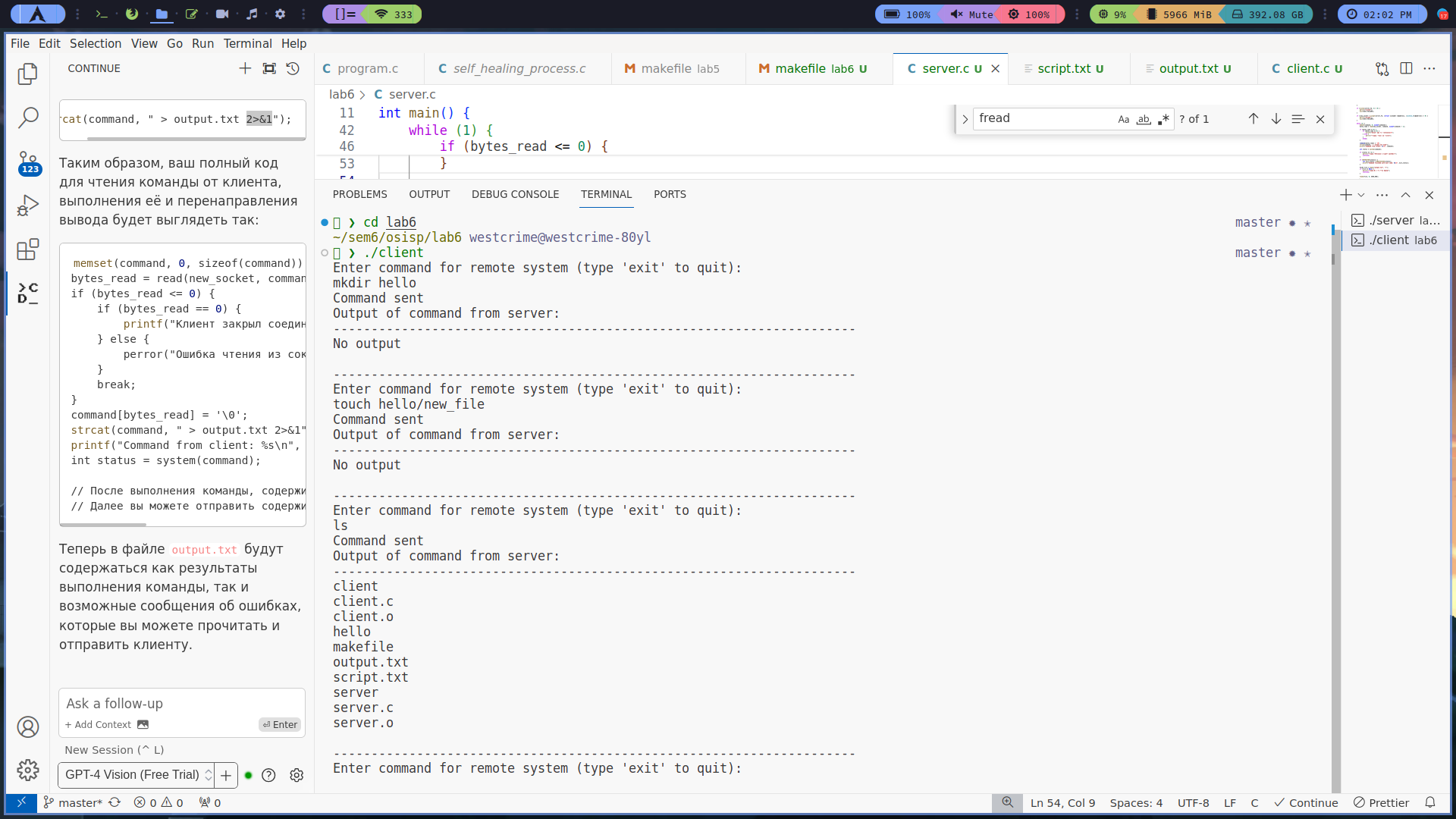


Рисунок 2 – Пример работы клиента

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Процессы UNIX [Электронный ресурс]. – Электронные данные. – Режим доступа: http://kharchuk.ru/home/15-unix-foundations/80-unix-processes

[2] system() in C [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.geeksforgeeks.org/system/

ПРИЛОЖЕНИЕ А

**(обязательное)   
Листинг кода**

server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/wait.h>

#define PORT 8081

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

ssize\_t bytes\_read;

char command[2048] = {0};

char answer[2048] = {0};

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

while (1) {

memset(command, 0, sizeof(command));

bytes\_read = read(new\_socket, command, sizeof(command) - 1);

if (bytes\_read <= 0) {

if (bytes\_read == 0) {

printf("Клиент закрыл соединение\n");

} else {

perror("Ошибка чтения из сокета");

}

break;

}

command[bytes\_read] = '\0';

strcat(command, " > output.txt 2>&1");

printf("Command from client: %s\n", command);

int status = system(command);

if (status == -1) {

perror("Ошибка выполнения команды system()");

continue;

}

if (WIFEXITED(status)) {

int exit\_status = WEXITSTATUS(status);

printf("Command finished with exit code: %d\n", exit\_status);

}

FILE\* file = fopen("output.txt", "r");

if (file == NULL) {

perror("Ошибка при открытии файла");

continue;

}

fseek(file, 0, SEEK\_END);

long fileSize = ftell(file);

rewind(file);

char \*content = malloc(fileSize + 1);

if (content == NULL) {

perror("Ошибка при выделении памяти");

fclose(file);

continue;

}

size\_t bytesRead = fread(content, 1, fileSize, file);

if (bytesRead < fileSize) {

perror("Ошибка при чтении файла");

free(content);

fclose(file);

continue;

}

content[bytesRead] = '\0';

if (strlen(content) == 0) {

strcpy(content, "No output\n");

}

fclose(file);

if (send(new\_socket, content, strlen(content), 0) == -1) {

perror("Ошибка при отправке данных");

} else {

printf("Output of command sent.\n");

}

free(content);

}

close(new\_socket);

close(server\_fd);

return 0;

}

client.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 8081

void print\_separator() {

printf("---------------------------------------------------------------------\n");

}

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char command[2048] = {0};

char buffer[2048] = {0};

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

while (1) {

printf("Enter command for remote system (type 'exit' to quit):\n");

fgets(command, sizeof(command), stdin);

command[strcspn(command, "\n")] = 0;

if (strcmp(command, "exit") == 0) {

break;

}

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

perror("send");

break;

}

printf("Command sent\n");

ssize\_t bytes\_received = read(sock, buffer, sizeof(buffer) - 1);

if (bytes\_received <= 0) {

if (bytes\_received == 0) {

printf("Server closed the connection\n");

} else {

perror("read");

}

break;

}

buffer[bytes\_received] = '\0';

printf("Output of command from server:\n");

print\_separator();

printf("%s\n", buffer);

print\_separator();

}

close(sock);

return 0;

}