

Москва, 2021

Часть 1

Разработать клиент-серверное приложение с использованием сетевых сокетов и протокола TCP для передачи бинарных файлов от сервера клиентам. Сервер должен позволять подключение нескольких клиентов. Имя файла задается клиентом при подключении в виде текстовой строки. Сервер должен отобразить, используя пользовательский интерфейс, названия всех файлов в папке с исполняемым файлом сервера, предоставив клиенту выбор.

В файле includes.hpp определены основные константы.

includes.hpp

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fstream>
#include <errno.h>
#include <dirent.h>
#include <limits.h>

#define HOST "localhost"
#define PORT 3000
#define NUM 5
#define SIZE 100
#define BLOCK 1024
```

Сервер

server.cpp

```
#include "includes.hpp"

int clients_arr[NUM];

static void list_file(const char *filename, char buf[]) {
    struct dirent *dirp;
    DIR *dp;
    int i = 1, count = 0;

    if ((dp = opendir(filename)) == NULL) {
        printf("ERROR: opendir %s\n", filename);
        return;
    }

    while ((dirp = readdir(dp)) != NULL) {
        if (!strcmp(dirp->d_name, ".") || !strcmp(dirp->d_name, ".."))
            continue;
        count += snprintf(buf + count, sizeof(dirp->d_name), "%s\n", dirp->d_name);
    }

    if (closedir(dp) < 0)
        printf("ERROR: close catalog %s\n", filename);
}

void send_file(char filename[], int sock) {
    FILE *f;
    struct stat f_info;
    int size, num = -1, temp;
    char buff[BLOCK];

    if ((f = fopen(filename, "rb")) == NULL) {
        printf("Cannot open file\n");
        send(sock, &num, sizeof(int), 0);
        return;
    }

    fstat(fileno(f), &f_info);
    size = f_info.st_size;
    num = size / BLOCK + 1;
    send(sock, &num, sizeof(int), 0);
```

```

    for(int i = 0; i < num; i++){
        temp = fread(buff, 1, BLOCK, f);
        send(sock, buff, temp, 0);
    }
}

void close_socket(int sock){
    for (int i = 0; i < NUM; i++)
        if (clients_arr[i])
            close(clients_arr[i]);
    close(sock);
}

void create_connection(int sock, char buf[]) {
    struct sockaddr_in client_addr;
    int addrSize = sizeof(client_addr), flag = 1;

    int new_sock = accept(sock, NULL, NULL);
    if (new_sock < 0) {
        perror("ERROR: accept failed");
        exit(-1);
    }

    printf("New connection added. Client fd = %d\t ip = %s:%d\n",
new_sock, inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    if (send(new_sock, buf, strlen(buf), 0) < 0){
        close(new_sock);
        perror("ERROR: send() failed");
        exit(-1);
    }

    for (int i = 0; i < NUM && flag; i++){
        if (!clients_arr[i]) {
            clients_arr[i] = new_sock;
            flag = 0;
        }
    }
}

int process_customers(int sock, char lst_buf[]){
    fd_set set;
    char buf[SIZE];
    int max_fd, err, rsize, fd;

```

```
while (1) {
    struct timeval interval = {30, 0};

    FD_ZERO(&set);
    FD_SET(sock, &set);

    max_fd = sock;

    for (int i = 0; i < NUM; i++){
        if (clients_arr[i] > 0)
            FD_SET(clients_arr[i], &set);

        if (clients_arr[i] > max_fd)
            max_fd = clients_arr[i];
    }

    err = select(max_fd + 1, &set, NULL, NULL, &interval);
    if (err < 0){
        close_socket(sock);
        perror("ERROR: select failed");
        return EXIT_FAILURE;
    }
    else if (!err){
        close_socket(sock);
        printf("Time of waiting is over\n");
        return 0;
    }
    if (FD_ISSET(sock, &set))
        create_connection(sock, lst_buf);

    for (int i = 0; i < NUM; i++){
        fd = clients_arr[i];
        if ((fd > 0) && FD_ISSET(fd, &set)){
            rsize = recv(fd, buf, sizeof(buf), 0);
            if (!rsize){
                printf("Client was disconnected\n");
                clients_arr[i] = 0;
            }
            else {
                buf[rsize] = '\0';
                printf("Server got: %s\n", buf);

                send_file(buf, fd);
            }
        }
    }
}
```

```

    }
}
}

int init_server(){
    struct sockaddr_in serv_addr;

    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("ERROR: socket failed");
        return EXIT_FAILURE;
    }
    fcntl(sock, F_SETFL, O_NONBLOCK);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        close(sock);
        perror("ERROR: bind failed");
        return EXIT_FAILURE;
    }
    if (listen(sock, NUM) < 0){
        close(sock);
        perror("ERROR: listen failed");
        return EXIT_FAILURE;
    }
    return sock;
}

int main(){
    int err;
    int server = init_server();
    char buf[256];
    printf("Server was created\n");
    list_file("./", buf);
    err = process_customers(server, buf);
    close_socket(server);
    printf("Server was closed\n");
    return 0;
}

```

Клиент

client.cpp

```
#include "includes.hpp"

int init_client(){
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0){
        perror("ERROR: socket() failed");
        exit(-1);
    }
    return sock;
}

void connect_client(int client){
    struct sockaddr_in serv_addr;
    struct hostent *server;

    server = gethostbyname(HOST);
    if (!server) {
        close(client);
        perror("ERROR: gethostbyname() failed");
        exit(-1);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr = *((struct in_addr*) server->h_addr_list[0]);
    serv_addr.sin_port = htons(PORT);
    if (connect(client, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0){
        close(client);
        perror("ERROR: connect() failed");
        exit(-1);
    }
}

int get_lst(int client){
    int rsize, err = EXIT_FAILURE;
    char lst_buf[256];

    rsize = recv(client, lst_buf, sizeof(lst_buf), 0);
    if (rsize){
        lst_buf[rsize] = '\0';
        printf("Client got:\n%s\n", lst_buf);
        return EXIT_SUCCESS;
    }
}
```

```

    }
    printf("Server was disconnected\n");

    return EXIT_FAILURE;
}

int choose_file(int client, char buf[])
{
    printf(">>> ");
    scanf("%s", buf);

    if (send(client, buf, strlen(buf), 0) < 0){
        close(client);
        perror("ERROR: send() failed");
        return EXIT_FAILURE;
    }
    printf("Client sent: %s\n", buf);
    return EXIT_SUCCESS;
}

int get_num_blocks(int client){
    int rsize, num;

    rsize = recv(client, &num, sizeof(int), 0);
    if (num < 0){
        printf("File was not found\n");
        return num;
    }
    return num;
}

void get_file(int client, char filename[]){
    int num, rsize;
    char buf[BLOCK + 1];
    FILE *f;

    if ((num = get_num_blocks(client)) < 0) return;

    f = fopen(filename, "wb");

    for (int i = 0; i < num; i++){
        rsize = recv(client, buf, BLOCK, 0);
        buf[rsize] = '\0';
        fwrite(&buf, 1, rsize, f);
    }
}

```



```

    }
    printf("File was downloaded\n");

    fclose(f);
}

void do_task(int client){
    char buf[100];

    if (get_lst(client)) return;
    if (choose_file(client, buf)) return;

    get_file(client, buf);
}

int main()
{
    int client = init_client();
    connect_client(client);
    do_task(client);

    return 0;
}

```

Демонстрация работы

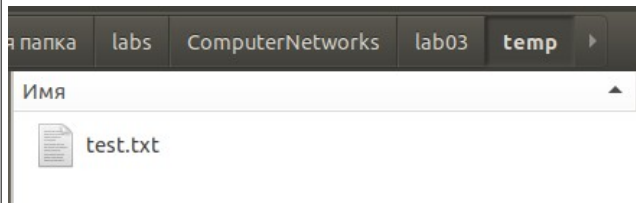
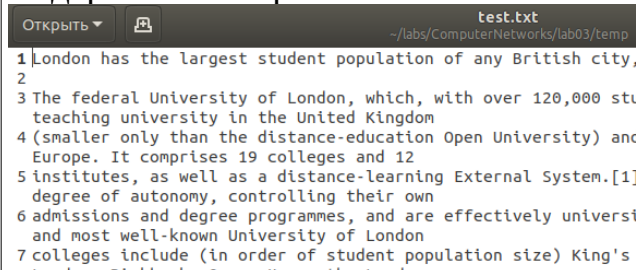
Работа сервера

```

ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/labs/ComputerNetworks/lab03/par
t1$ ./server
Server was created
New connection added. Client fd = 4      ip = 0.0.0.0:0
Server got: test.txt
Client was disconnected
New connection added. Client fd = 6      ip = 214.85.0.0:11256
Server got: 12345
Cannot open file
Client was disconnected
close_socket
Time of waiting is over
close_socket
Server was closed

```

Работа клиента

Действия клиента	Результат
<pre>ekaterina@ekaterina-HP-470-G7-N emp\$../part1/client Client got: client server.cpp server test.txt includes.hpp client.o client.cpp .vscode server.o Makefile >>> test.txt Client sent: test.txt File was downloaded</pre>	<p>В папке temp (в который выполнялись действия клиента) появился файл test.txt</p>  <p>Содержимое сохранено.</p> 
<pre>ekaterina@ekaterina-HP-470-G7-N emp\$../part1/client Client got: client server.cpp server test.txt includes.hpp client.o client.cpp .vscode server.o Makefile >>> 12345 Client sent: 12345 File was not found</pre>	<p>В папке temp (в который выполнялись действия клиента) файл не появился, так как такого файла не существует.</p>

Часть 2

Разработать http сервер для обработки GET-запросов и предоставления статической информации клиенту. Разработать http-клиент для проверки данного сервера с помощью GET-запросов. Использовать системные сокеты и транспортный протокол TCP. Язык C/C++. Возможно использование ранее разработанного TCP-сервера в качестве основы.

Выполнить дополнительную задачу в зависимости от варианта.

Дополнительная задача:

- ➔ Сохранять информацию обо всех посещенных пользователем страницах с указанием имени/идентификатора пользователя

В программе был реализован многопоточный сервер с обработкой соединений через использование динамически распределяемого пула потоков + выполнение доп. задачи.

includes.hpp для клиента

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fstream>
#include <errno.h>
#include <dirent.h>
#include <limits.h>

#define HOST "127.0.0.1"
```

```
#define PORT 8001
#define SIZE 1024
#define REQUEST "test.txt"
```

client.cpp

```
#include "includes.hpp"

int init_client(){
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock < 0){
        perror("ERROR: socket() failed");
        exit(-1);
    }
    return sock;
}

void connect_client(int client){
    struct sockaddr_in serv_addr;
    struct hostent *server;

    server = gethostbyname(HOST);
    if (!server) {
        close(client);
        perror("ERROR: gethostbyname() failed");
        exit(-1);
    }

    serv_addr.sin_family = PF_INET;
    serv_addr.sin_addr = *((struct in_addr*) server->h_addr_list[0]);
    serv_addr.sin_port = htons(PORT);
    if (connect(client, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0){
        close(client);
        perror("ERROR: connect() failed");
        exit(-1);
    }
}

std::string form_request(std::string path){
    const std::string vrs = "HTTP/1.1";
    return "GET /" + path + " " + vrs + "\r\n";
}

int do_task(int client){
```

```

char buf[SIZE];
std::string request = form_request(REQUEST);

if (send(client, request.c_str(), request.length(), 0) < 0){
    close(client);
    perror("ERROR: send() failed");
    return EXIT_FAILURE;
}

if (recv(client, buf, sizeof(buf), 0) < 0){
    perror("ERROR: recv() failed");
    return EXIT_FAILURE;
}

printf("Client has recieved an answer:\n\n%s", buf);
close(client);
}

int main(){
    int client = init_client();
    connect_client(client);
    do_task(client);
    return 0;
}

```

includes.hpp для сервера

```

#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fstream>
#include <errno.h>
#include <dirent.h>
#include <limits.h>
#include <iostream>

```

```
#include <map>

#include "threads.hpp"

#define HOST "127.0.0.1"
#define PORT 8001
#define NUM 5
#define SIZE 100
#define BASE_PATH "./"
#define HISTORY_REQUESTS "./history.txt"
```

server.cpp

```
#include "includes.hpp"

int clients_arr[NUM];
std::string addr_arr[NUM];

void save_info(const std::string user, const std::string &path){
    std::ofstream f(HISTORY_REQUESTS, std::ios::app);
    f << user << "\t" << path << "\n";
}

void get_params(const std::string buf, std::string &kind, std::string &path,
std::string &vrs, std::string &ext){
    int from = 0, to = buf.find(" ");

    kind = buf.substr(from, to - from);
    from = to + 1;
    to = buf.find(" ", from);

    path = buf.substr(from, to - from);
    path = path.substr(1);

    from = path.find(".");
    ext = path.substr(from + 1, path.length());

    from = to + 1;
    to = buf.find("\r", from);

    vrs = buf.substr(from, to - from);
}

void form_response(const std::string buf, int fd, std::string ip){
```

```

std::string kind, path, vrs, ext;
get_params(buf, kind, path, vrs, ext);

if (kind != "GET"){
    perror("ERROR: only GET can be processed");
    return;
}
if (vrs != "HTTP/1.1"){
    perror("ERROR: only HTTP/1.1 can be processed");
    return;
}

std::ifstream file(BASE_PATH + std::string(path));
std::string response = "";
std::string content_type, body = "", status, status_code;
std::string temp;

std::map<std::string, std::string> type_arr = {
    { "html", "text/html" },
    { "jpg", "image/jpeg" },
    { "jpeg", "image/jpeg" },
    { "png", "image/png" },
    { "txt", "text/html" },
};

content_type = type_arr[ext];
if (content_type == "")
    content_type = "text/html;";
if (file.is_open()){
    while (std::getline(file, temp))
        body += temp + "\n";

    status_code = "200";
    status = "OK";
}
else{
    status_code = "404";
    status = "Not Found";
    body = "<html>\n\r<body>\n\r<h1>404 Not Found</h1>\n\r</body>\n\r</html>";
}

response += vrs + " " + status_code + " " + status + "\r\n";
response += "Content-Length: " + std::to_string(body.length()) + "\r\n";

```

```

response += "Content-Type: " + content_type + "\r\n";
response += "Connection: closed\r\n\r\n";
response += body;

save_info(ip, path);
send(fd, response.c_str(), response.size(), 0);
}

void close_socket(int sock){
    for (int i = 0; i < NUM; i++)
        if (clients_arr[i])
            close(clients_arr[i]);
    close(sock);
}

void create_connection(int sock){
    struct sockaddr_in client_addr;
    socklen_t size_client = sizeof(client_addr);
    int flag = 1;

    int new_sock = accept(sock, (struct sockaddr*) &client_addr, &size_client);
    if (new_sock < 0) {
        perror("ERROR: accept failed");
        exit(-1);
    }

    printf("New connection added. Client fd = %d\t ip = %s:%d\n",
        new_sock, inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    for (int i = 0; i < NUM && flag; i++){
        if (!clients_arr[i]) {
            clients_arr[i] = new_sock;
            addr_arr[i] = inet_ntoa(client_addr.sin_addr) + std::string(":") +
std::to_string(ntohs(client_addr.sin_port));
            flag = 0;
        }
    }
}

int process_customers(int sock, ThreadPool &pool){
    fd_set set;
    std::string buf;
    char buf_arr[SIZE];
    int max_fd, err, rsize, fd;

```



```

while (1){
    struct timeval interval = {30, 0};

    FD_ZERO(&set);
    FD_SET(sock, &set);

    max_fd = sock;

    for (int i = 0; i < NUM; i++){
        if (clients_arr[i] > 0)
            FD_SET(clients_arr[i], &set);

        if (clients_arr[i] > max_fd)
            max_fd = clients_arr[i];
    }

    err = select(max_fd + 1, &set, NULL, NULL, &interval);
    if (err < 0){
        close_socket(sock);
        perror("ERROR: select failed");
        return EXIT_FAILURE;
    }
    else if (!err){
        close_socket(sock);
        printf("Time of waiting is over\n");
        return 0;
    }
    if (FD_ISSET(sock, &set))
        create_connection(sock);

    for (int i = 0; i < NUM; i++){
        fd = clients_arr[i];
        if ((fd > 0) && FD_ISSET(fd, &set)){
            rsize = recv(fd, buf_arr, sizeof(buf_arr), 0);
            if (!rsize)
                printf("Client was disconnected\n");
            else if (rsize == -1){
                perror("ERROR: recv failed");
                return EXIT_FAILURE;
            }
        }
        else{
            buf_arr[rsize] = '\0';
            buf = buf_arr;
        }
    }
}

```

```

        auto res = pool.add(form_response, buf, fd, addr_arr[i]);
        res.get();
        close(fd);
    }

    clients_arr[i] = 0;
    addr_arr[i] = "";
}
}

return EXIT_SUCCESS;
}

int init_server(){
    struct sockaddr_in serv_addr;

    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("ERROR: socket failed");
        return EXIT_FAILURE;
    }
    fcntl(sock, F_SETFL, O_NONBLOCK);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        close(sock);
        perror("ERROR: bind failed");
        return EXIT_FAILURE;
    }

    if (listen(sock, NUM) < 0){
        close(sock);
        perror("ERROR: listen failed");
        return EXIT_FAILURE;
    }
    return sock;
}

int main(){
    int err;
    int server = init_server();

```

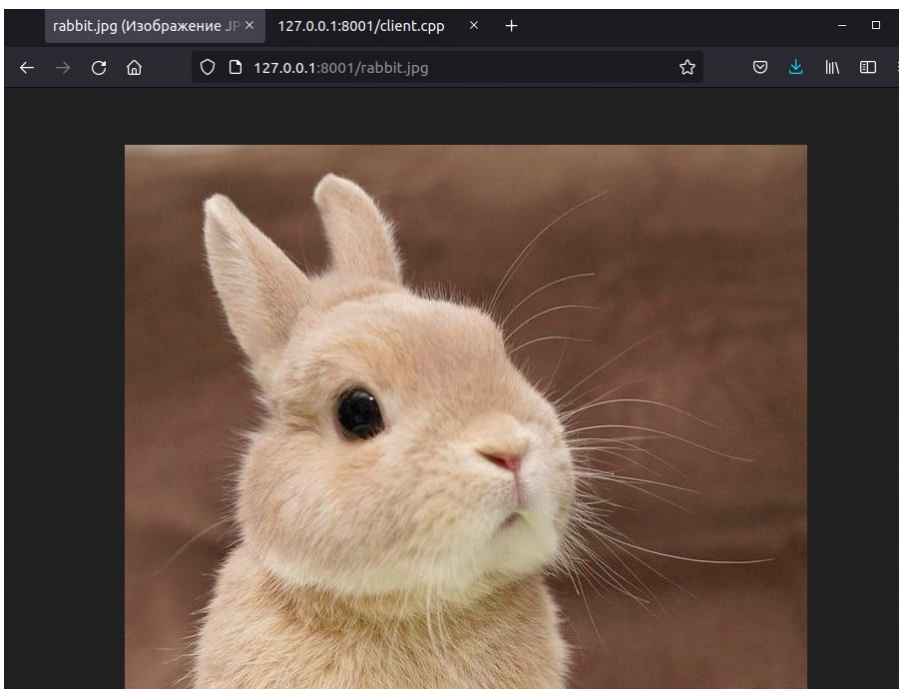
```
ThreadPool pool(NUM);

printf("Server was created\n");
err = process_customers(server, pool);
close_socket(server);
printf("Server was closed\n");
return err;
}
```

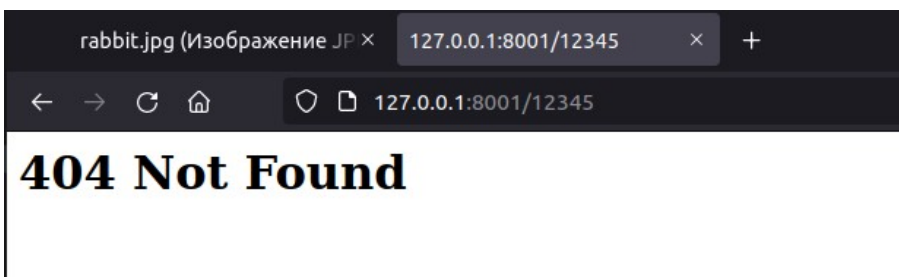
Демонстрация работы:

Клиент

1. файл существует



2. файл не существует



Сервер

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/labs/ComputerNetwork
s/lab03/part2/server$ ./server
Server was created
New connection added. Client fd = 4      ip = 127.0.0.1:49134
New connection added. Client fd = 4      ip = 127.0.0.1:49136
Time of waiting is over
Server was closed
```

Из консоли:

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/labs/ComputerNetwork
s/lab03/part2/client$ ./client
Client has recieved an answer:

HTTP/1.1 200 OK
Content-Length: 2409
Content-Type: text/html
Connection: closed

London has the largest student population of any British city, a
lthough not the highest per capita.

The federal University of London, which, with over 120,000 stude
nts, is the largest contact teaching university in the United Ki
ngdom
(smaller only than the distance-education Open University) and o
```

Дополнительная задача

```
1 127.0.0.1:49026 rabbit.jpg
2 127.0.0.1:49032 favicon.ico
3 127.0.0.1:49034 test.txt
4 127.0.0.1:49036 test11.txt|
5 127.0.0.1:49056 test11.txt
6 127.0.0.1:49238 test11.txt
7 127.0.0.1:49240 test.txt
8 127.0.0.1:49432 rabbit.jpg
9 127.0.0.1:49434 test.txt
10 127.0.0.1:40844 test.txt
11 127.0.0.1:40848 test.txt
12 127.0.0.1:40850 favicon.ico
13 127.0.0.1:40852 test.txt
14 127.0.0.1:40854 test.txt
15 127.0.0.1:40856 test.txt
16 127.0.0.1:40858 test.txt
17 127.0.0.1:41258 test.txt
18 127.0.0.1:48936 test.txt
19 127.0.0.1:48990 rabbit.jpg
20 127.0.0.1:48992 favicon.ico
21 127.0.0.1:48994 test.txt
22 127.0.0.1:48996 test.txt
23 127.0.0.1:48998 client.cpp
24 127.0.0.1:49126 rabbit.jpg
25 127.0.0.1:49128 client.cpp
26 127.0.0.1:49130 client.cpp
27 127.0.0.1:49134 rabbit.jpg
28 127.0.0.1:49136 12345
29 127.0.0.1:49146 test.txt
```