Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №8

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Студенты: Суслов А.В.

Эльвейн Н.Л.  
 Олимпиади М.Б.

Преподаватели: Хайленко Е.А.  
 Филиппова Е.В.

Вариант: 1

1. **Цель работы**

Практическое освоение механизма сокетов. Построение TCP-соединений

для межпроцессного взаимодействия программ Клиента и Сервера в модели

“клиент-сервер”.

1. **Задание**

Эмуляция DNS сервера. Клиент подсоединяется к серверу, IP которого хранится в файле dns.url, и делает ему запрос на подключение к серверу "Имя сервера". DNS-сервер имеет список, хранящийся в файле о соответствии имен серверов и IP-адресов. Если в списке нет "имени сервера", запрошенного клиентом, то сервер DNS подключается последовательно к другим серверам, хранящимся в файле dns.url, и т.д. Если сервер не найден, клиенту возвращается соответствующее сообщение.

1. **Описание использованных системных вызовов**

**int socket(int domain, int type, int protocol) <sys/socket.h>**

Создаёт новый сокет типа type в домене domain, основанный на протоколе protocol. Возвращает файловый дескриптор нового сокета или -1 при ошибке.

**int connect(int fd, CONST\_SOCKADDR\_ARG addr, socklen\_t len) <sys/socket.h>**

Открывает соединение на сокете fd к узлу addr (длина адреса которого len байт). Для сокетов без установления соединения (вроде UDP) устанавливает адрес по умолчанию для отправки и отбрасывает входящие пакеты не с этого адреса. Возвращает 0 при успехе, либо -1 при ошибке.

**int bind(int fd, CONST\_SOCKADDR\_ARG addr, socklen\_t len) <sys/socket.h>**

Назначает сокету fd локальный адрес addr (длина адреса которого len байт).

**int listen(int fd, int n) <sys/socket.h>**

Запускает режим прослушивания на сокете fd. N одновременных запросов на соединение будут ставиться в очередь, дальнейшие будут отклоняться.

Возвращает 0 при успехе, либо -1 при ошибке.

**int accept(int fd, SOCKADDR\_ARG addr, socklen\_t \*\_\_restrict addr\_len) <sys/socket.h>**

Ожидает входящих соединений на сокете fd. Когда происходит соединение, для него создаётся новый сокет, в addr записывается адрес подключившегося клиента (длина адреса которого addr\_len байт). Возвращает дескриптор нового сокета, либо -1 при ошибке.

**ssize\_t read(int fd, void \*buf, size\_t n) <unistd.h>**

Выполняет чтение из файла по дескриптору fd не более n байт в память buf. Возвращает количество фактически прочитанных байт. Тип ssize\_t эквивалентен short.

**ssize\_t write(int fd, const void \*buf, size\_t n) <unistd.h>**

Записывает в файл по дескриптору fd не более n байт из памяти buf. Возвращает количество фактически записанных байт.

**int close(int fd) <unistd.h>**

Закрывает дескриптор файла. Фактически это может быть как файл, так и канал.

1. **Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdexcept> // for exceptions

#pragma region Global vars

#define BUFSIZE 256

#define SOCKET\_QUEUE 10

#define SV\_PORT 2000

const char \*\_server\_file = "dns.url";

const char \*\_resolv\_file = "hosts";

char buf[BUFSIZE];

#define inet\_atoa(in\_addr\_t) \

inet\_ntop(AF\_INET,(void\*)&in\_addr\_t,buf+BUFSIZE-INET\_ADDRSTRLEN-1,INET\_ADDRSTRLEN)

FILE \*logfile = stderr;

bool server\_mode = false;

#pragma endregion

typedef enum { MT\_DATA, MT\_ERROR } msg\_t; // Типы сообщений

typedef struct { // client-server message template

msg\_t type;

char str[BUFSIZE];

} data\_t;

// Connect socket to remote address using its IP and port.

int connect\_to(int sockfd, in\_addr\_t ip\_addr, ushort port) {

sockaddr\_in s\_addr; // структура с информацией о сервере

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

s\_addr.sin\_port = htons(port);

s\_addr.sin\_addr.s\_addr = ip\_addr;

return connect(sockfd, (sockaddr\*)&s\_addr, sizeof(s\_addr));

}

// Bind socket locally to become a server.

int bind\_to(int sockfd, ushort port, in\_addr\_t addr = INADDR\_ANY) {

sockaddr\_in s\_addr; // структура с информацией о сервере

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

s\_addr.sin\_addr.s\_addr = addr; // 0.0.0.0 by default

s\_addr.sin\_port = htons(port);

return bind(sockfd, (sockaddr\*)&s\_addr, sizeof(s\_addr));

}

// Read single row from server list and return ready-to-use IP and port.

bool get\_server(FILE \*f, in\_addr\_t &ip, ushort &port) {

if (f == NULL) return false; // если нет файла

char addr[BUFSIZE];

if (fscanf(f, "%s %hd", addr, &port) < 2) // %hd means s**h**ort integer

return false;

in\_addr ip\_addr;

inet\_aton(addr, &ip\_addr); // функция перевода ip-адреса в нужный тип

ip = ip\_addr.s\_addr; // take only IP value

return true;

}

// Lookup for host\_s in local resolv file.

bool find\_ip(const char \*host\_s, char \*addr\_s, const char \*filename = \_resolv\_file) {

FILE \*f = fopen(filename, "r");

if (f == NULL) { // если нет файла

fprintf(logfile, "[!] Cannot open resolv file\n");

return false;

}

char hostname[BUFSIZE], addr[BUFSIZE];

while (fscanf(f, "%s %s", hostname, addr) == 2) { // пока не конец файла

if (strcmp(host\_s, hostname) == 0) { // hostnames match

strncpy(addr\_s, addr, BUFSIZE - 1);

return true;

}

}

fprintf(logfile, "[i] IP address for hostname '%s' has not found locally\n", host\_s);

fclose(f);

return false;

}

// Query all known `DNS` servers to search for hostname in their databases.

bool query\_external\_dns(const char \*hostname, char \*addr, const char \*server\_file) {

FILE \*f = fopen(server\_file, "r");

if (f == NULL) {

fprintf(logfile, "[!] Server file '%s' couldn't be opened.\n", server\_file);

return false;

}

in\_addr\_t dns\_ip;

u\_short dns\_port;

int sockfd;

while (get\_server(f, dns\_ip, dns\_port)) { // получаем ip-адрес вышестоящего сервера

try {

if ((sockfd = socket(PF\_INET, SOCK\_STREAM, SOL\_IP)) == -1)

throw std::runtime\_error("Could not create socket");

if (connect\_to(sockfd, dns\_ip, dns\_port) < 0) { // соединяемся с сервером

sprintf(buf, "Could not connect to server '%s:%d'", inet\_atoa(dns\_ip), dns\_port);

throw std::runtime\_error(buf);

}

data\_t msg;

msg.type = MT\_DATA;

strcpy(msg.str, hostname); // формируем сообщение

write(sockfd, &msg, sizeof(msg)); // отправляем сообщение серверу

read(sockfd, &msg, sizeof(msg)); // получаем сообщение сервера; blocking mode

if (msg.type == MT\_DATA) {

strncpy(addr, msg.str, BUFSIZE - 1);

return true;

}

}

catch (std::runtime\_error e) {

fprintf(logfile, "[!] %s\n", e.what());

}

close(sockfd); // finally

}

return false;

}

// Perform DNS server behaviour, which may be as authoritative as forwarder.

int perform\_server(ushort port = SV\_PORT, const char \*resolv\_file = \_resolv\_file, const char \*dns\_file = \_server\_file) {

data\_t msg;

int sockfd;

try {

if ((sockfd = socket(PF\_INET, SOCK\_STREAM, SOL\_IP)) < 0)

throw std::runtime\_error("Cannot create socket");

if (bind\_to(sockfd, port) < 0) // связываем сокет с ip-адресом и портом

throw std::runtime\_error("Error in bind");

if (listen(sockfd, SOCKET\_QUEUE) < 0) // открываем доступ для клиентов

throw std::runtime\_error("Could not set listen mode");

fprintf(logfile, "[i] Server started on port %d\n", port);

}

catch (std::runtime\_error e) {

fprintf(logfile, "[x] Server: %s\n", e.what());

close(sockfd);

return EXIT\_FAILURE;

}

int cl\_sockfd; // ждем подключений клиентов, устанавливаем соединение

while ((cl\_sockfd = accept(sockfd, NULL, NULL)) > 0) {

read(cl\_sockfd, &msg, sizeof(msg)); // получаем сообщение клиента

msg.type = MT\_DATA;

if (find\_ip(msg.str, msg.str, resolv\_file)) { // если нашли заданное имя в таблице

}

else if (query\_external\_dns(msg.str, msg.str, dns\_file)) { // если на другом сервере

}

else { // если не нашли

msg.type = MT\_ERROR;

memset(msg.str, '\0', BUFSIZE);

}

write(cl\_sockfd, &msg, sizeof(msg)); // посылаем сообщение клиенту

close(cl\_sockfd); // закрываем дескриптор на сокет клиента

}

close(sockfd); // закрываем дескриптор на сокет сервера

return EXIT\_SUCCESS;

}

// Perform DNS client behaviour.

int perform\_client(const char \*hostname = "ultibot.ru", const char \*server\_file = \_server\_file) {

char addr[BUFSIZE];

if (query\_external\_dns(hostname, addr, server\_file)) {

printf("Server '%s' has ip '%s'\n", hostname, addr);

return EXIT\_SUCCESS;

}

else {

printf("No records for server name '%s'\n", hostname);

return EXIT\_FAILURE;

}

}

// Main entry point of application.

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

char \*host = NULL;

ushort port;

int e;

while ((e = getopt(argc, argv, ":c:d:")) != -1) {

switch (e) {

case 'c': host = optarg; break;

case 'd': server\_mode = true; host = optarg; break;

case '?': fprintf(logfile, "[!] Unknown command-line argument with value: %s\n", optarg); exit(1); break;

}

};

if (server\_mode) {

setbuf(logfile, NULL);

if (host != NULL && sscanf(host, "%hd", &port) == 1)

return perform\_server(port);

else // this 'else' left for readability

return perform\_server();

}

if (host != NULL)

return perform\_client(host);

else

return printf("Usage:\n-c <hostname>\t - resolve <hostname> using %s file\n"

"-d <port>\t - run as server on port <port>\n", \_server\_file);

}

1. **Тестирование**