1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «**Сетевые взаимодействия**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 23656/4 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. старший преподаватель Резединова Е.Ю.

<*подпись*>

1. Санкт-Петербург
2. 2019

Цели работы

Изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

Задачи работы

1. Реализовать TCP-client с блокирующими сокетами, отправляющий сообщения в нижеуказанном формате.

*dd.mm.yyyy dd.mm.yyyy hh:mm:ss Message*

Но предварительно послав 3 байта – коды символов ‘p’, ‘u’ и ‘t’ , как обозначение, что следующими пойдут байты сообщений. А именно:

* Сначала отправляется 4 байта - номер сообщения в исходном файле, целое 32-битное число в СЕТЕВОМ порядке байтов, индексация от 0.
* затем 1 байт - значение дня (1-31) первой даты,
* затем 1 байт - значение месяца (1-12) первой даты,
* затем 2 байта - unsigned short, год (0-9999), в СЕТЕВОМ порядке байтов,
* затем отправляются 4 байта (1+1+2) второй даты в таком же виде,
* затем 1 байт - значение часов (hh, 0-23),
* затем 1 байт - значение минут (mm, 0-59),
* затем 1 байт - значение секунд (ss, 0-59),
* затем N+1 байт - N символов сообщения Message, а в конце передается дополнительно нулевой байт (как для конца строки) - маркер конца сообщения.

1. Реализовать TCP-сервер, работающий на неблокирующих сокетах и использующий механизм параллельного обслуживания select. Такой сервер должен отсылать ok на приходящие клиентские сообщения, и завершаться, если какой-либо клиент присылал stop. Все полученные сообщения распечатываются в msg.txt.
2. Реализовать UDP-клиент, использующий блокирующие сокеты. Такой клиент должен на каждую непустую строку формировать сообщение-дейтаграмму и отправлять ее на удаленный сервер. Дейтаграмма содержит данные в таком же формате, в каком tcpclient передавал на удаленный TCP-сервер. Клиент должен повторять отправку дейтаграмм с сообщениями до тех пор, пока на сервер не будут загружены хотя бы 20 ЛЮБЫХ сообщений из исходного файла
3. Реализовать UDP-сервер , работающий на неблокирующих сокетах и использующих механизм poll для параллельного обслуживания клиентов. Он принимает клиентов на заданном диапазоне портов и осуществляет проверку на повторность. Сервер завершается , если придёт stop от какого-либо клиента.

Ход работы

1. Описание алгоритма параллельного обслуживания клиентов для программы-сервера.
2. TCP-сервер(select)

Создаётся прослушивающий сокет и он привязывается к определенным адресам локального компьютера, потом он начинает прослушивание. Массив структур хранит информацию о сообщениях , присланных каждым клиентом. В бесконечном цикле используем select. Данная функция (select) позволяет процессу сообщить ядру, что нужно подождать наступления некоторых событий, и вывести процесс из состояния ожидания только после того, как наступит одно из таких событий или пройдет заданное количество времени.Аргументы readfds, определяют список дескрипторов, которые ядро должно проверить на возможность чтения, записи и на наличие исключений, соответственно. Список дескрипторов, как правило, это массив целых чисел, и каждое такое число рассматривается как bitset, то есть в массиве из 4 32-разрядных чисел мы сможем хранить следующую информацию:

fds[0]; // биты соответствуют дескрипторам от 0 до 31

fds[1]; // биты соответствуют дескрипторам от 31 до 63

fds[2]; // биты соответствуют дескрипторам от 64 до 95

fds[3]; // биты соответствуют дескрипторам от 96 до 127

Мы размещаем в памяти набор дескрипторов и с помощью нижеуказанных макросов работаем с ним:

void FD\_CLR(int fd, fd\_set \*set);

int FD\_ISSET(int fd, fd\_set \*set);

void FD\_SET(int fd, fd\_set \*set);

void FD\_ZERO(fd\_set \*set);

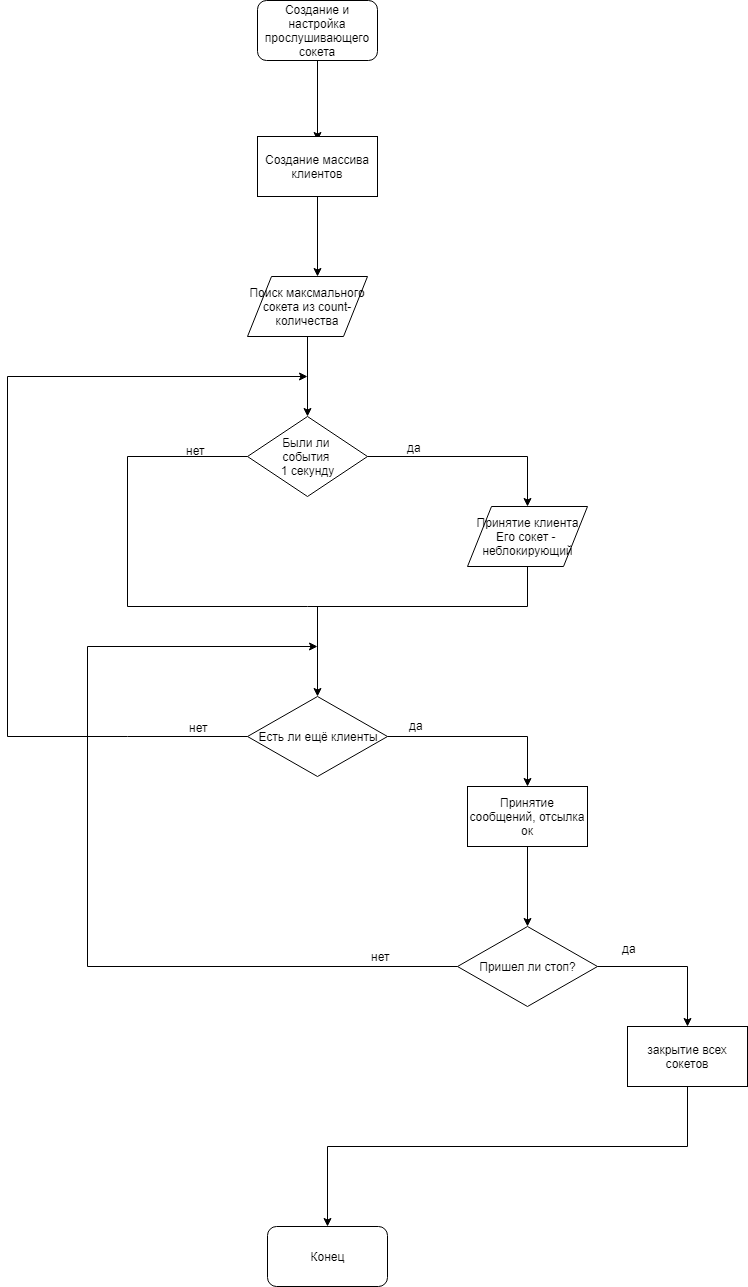
В процессе выполнения функция select изменяет значения, находящиеся в наборах дескрипторов. До того, как вызвать select мы указываем дескрипторы, которые нас интересуют, после ее выполнения получаем набор из дескрипторов, которые готовы. Проверить дескриптор можно с помощью макроса FD\_ISSET - если дескриптор не готов, то и соответствующий ему бит будет сброшен.

Структуру fd\_set необходимо формировать перед каждым вызовом select.

После того, проверки FD\_ISSET , accept возвращает новый сокет , который делаем неблокирующим , от появления новых советов будет изменятся граница цикла for для обработки поступающих сообщений.

В цикле for , в функции принятий сообщений отсылается ответ клиенту, и производится проверка на stop, и если текст сообщения оказался таковым, то закрываются все сокеты , в том числе и прослушивающий.

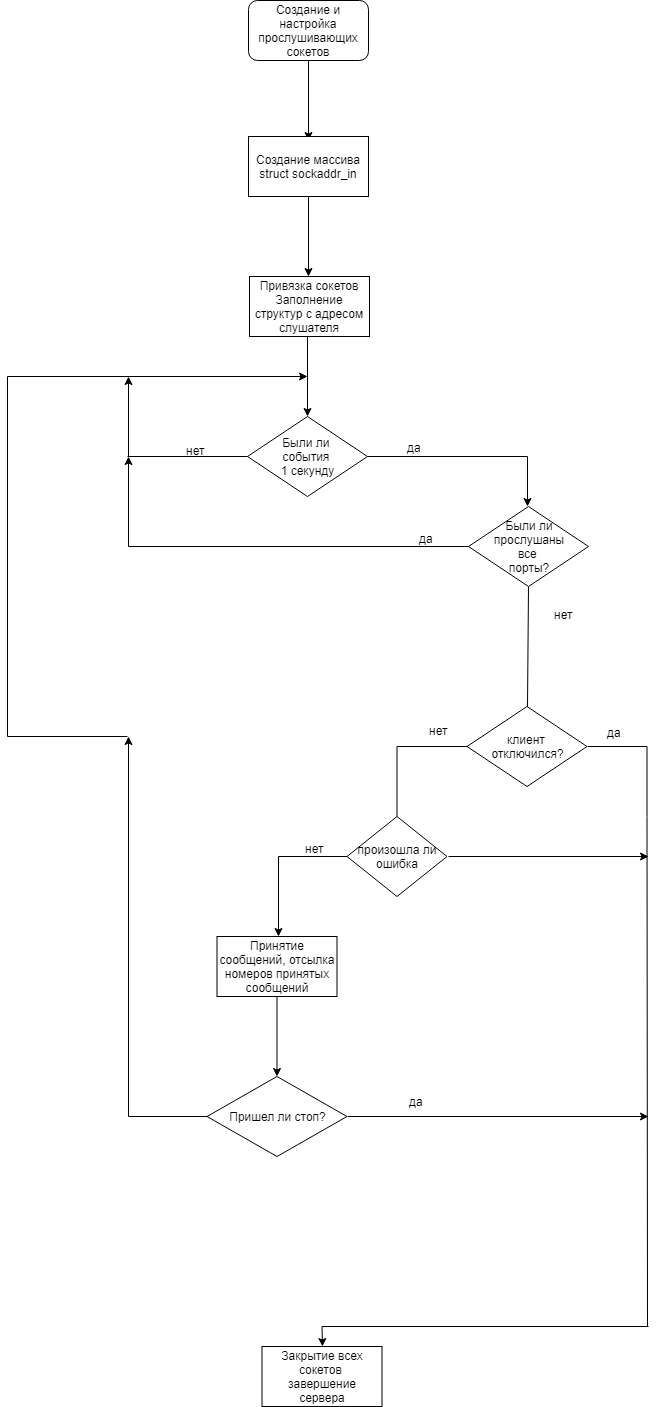
На следующей странице приведена блок-схема алгоритма



1. UDP-сервер(poll)

Был создан массив структур pollfd, такого размера, сколько портов прослушивается. На каждый из прослушиваемых портов был установлен слушающий сокет, каждый такой сокет переведён в неблокирующий режим. Заполнялся массив структур с адресом прослушивающих узлов.В отличие от структуры fd\_set, используемой в механизме select, массив структур poolfd не обязательно готовить каждый раз перед вызовом poll. Массив может быть заполнен один раз и передан в poll многократно. В бесконечном цикле не нужно вызывать функцию listen и accept для серверных UDP-сокетов. Если сокет доступен для чтения, то принимаем сообщения , отправляем 4 байта номер сообщения. Если текст сообщения – stop, закрываются все сокеты и завершается сервер.

Блок-схема приведена на следующей странице.



1. Описание применяемых в сервере структур данных и функций

а)TCP-сервер

Структура для организации информации о клиентах

struct clients

{

int s;

unsigned int ip;

};

Массив такого типа данных должен сохранять в себе сокет клиента и его ip.

Реализованные функции

|  |  |
| --- | --- |
| void resv\_msg(int, struct clients\*); | Принимает сообщения клиентов и сразу печатает их в файл , и отсылает ok клиенту на принятое сообщение |

b)UDP-сервер

Структура для организации параллельного обслуживания клиентов

Struct client

{

unsigned int ip;

unsigned short int port;

int idx[50];

}

Первое поле отвечает за уникальность клиента, второе – с кого порта принимали клиента(нужно для печати в файл), третье- для защиты от повторных записей в файл.

Реализованные функции

|  |  |
| --- | --- |
| void fill\_in(int num, struct sockaddr\_in \*addr) | Заполняет информацию об клиенте с номером num. |
| void add\_idx(int num, int idx) | Добавляет в структуру клиента с номером num информацию о том, что сообщение idx было получено |
| int send\_notice(int num, int s, sockaddr\_in \*addr) | Посылает на сокет s по адресу addr информацию о полученных от клиента num сообщениях |
| int recv\_msg(int num, int cs) | Получает сообщение от клиента cs под номером num, форматирует, пишет его в файл и отправляет уведомление |

1. Описание структуры кода и алгоритма программы-клиента
2. TCP-клиент

Программа создаёт TCP сокет, заполняет структуру с адресом удалённого компьютера и производит 10 попыток подключения с интервалом в 100 мс. В случае неудачи программа завершается с соответствующей ошибкой. Если же соединение удалось установить, клиент отправляет на сервер “put” и затем считывает построчно сообщения из исходного файла, форматирует их и так же отправляет. После каждого отправленного сообщения клиент ожидает подтверждение от сервера. Так повторяется, пока сообщения в файле не закончится, или ошибка не произойдёт.После успешной отправки всех сообщений и получения всех подтверждений клиент закрывает сокет и файл, затем завершается программа.

1. UDP-клиент

Программа создает UDP сокет, заполняет структуру с адресом удаленного компьютера и начинает отправку дейтаграмм, аналогичных сообщениям TCP клиента. После отправки дейтаграмм клиент ожидает ответа сервера в течение 100мс. Если ответ получен, то из него извлекаются номера полученных сервером сообщений, повторная отправка которых не требуется. Оставшиеся дейтаграммы отправляются до тех пор, пока от сервера не будет получено подтверждение о получении 20 дейтаграмм, или всех, если их общее число меньше.При получении такого подтверждения клиент закрывает сокеты, файл, и завершает работу.

1. Временная диаграмма взаимодействия сервера и клиента
2. TCP



1. UDP



1. Описание тестовых ситуаций
2. TCP
3. Проверка того, производит ли клиент 10 попыток при подключении к серверу
4. Проверка с пустым исходным файлом
5. Проверка, игнорируются ли пустые строки в исходном файле
6. Проверка подключения нескольких (10) клиентов к одному серверу
7. Проверка передачи длинного сообщения (больше миллиона символов)
8. Проверка корректной работы в паре с эмулятором клиента/сервера
9. UDP
10. Проверка того, что клиент не завершится, пока не сможет доставить нужное количество сообщений
11. Проверка параллельного обслуживаний клиентов с разных портов
12. Проверка длинного сообщения и большого количества сообщений
13. Проверка корректной работы в паре с эмулятором клиента/сервера.
14. Листинги

TCP-клиент

#ifdef \_WIN32

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

// Директива линковщику: использовать библиотеку сокетов

#pragma comment(lib, "ws2\_32.lib")

#else // LINUX

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <errno.h>

#include<unistd.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#endif

#include <stdio.h>

#include <string.h>

#include<stdlib.h>

#include<cctype>

#include <iostream>

using namespace std;

#define N 30000

int number\_of\_message=0;

struct my\_time

{

unsigned int tm\_myday1;

unsigned int tm\_mymonth1;

unsigned short tm\_myyear1;

unsigned int tm\_myhour1;

unsigned int tm\_myminutes1;

unsigned int tm\_mysec1;

unsigned int tm\_myday2;

unsigned int tm\_mymonth2;

unsigned short tm\_myyear2;

char\* message;

};

int sock\_err(const char\* function, int s)

{

int err;

#ifdef \_WIN32

err = WSAGetLastError();

#elseerr = errno;

#endif

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void parsing\_file(char\* name\_of\_file,struct my\_time \*data\_and\_message);

void send\_messages(struct my\_time\* data\_and\_message,int s);

int check(char\* buf)

{

int points=4;

int i=0;

int doublepoints=1;

if(!strcmp(buf,"\n") || !strcmp(buf," "))

return 0;

while(buf[i]!='\0')

{

if(buf[i]=='.')

points--;

if(buf[i]==':')

doublepoints--;

if(points==0 && doublepoints==0)

return 1;

i++;

}

}

/\* A utility function to reverse a string \*/

void reverse(char str[], int length)

{

int start = 0;

int end = length -1;

while (start < end)

{

swap(\*(str+start), \*(str+end));

start++;

end--;

}

}

// Implementation of itoa()

char\* itoa(int num, char\* str, int base)

{

int i = 0;

bool isNegative = false;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

if (num == 0)

{

str[i++] = '0';

str[i] = '\0';

return str;

}

// In standard itoa(), negative numbers are handled only with

// base 10. Otherwise numbers are considered unsigned.

if (num < 0 && base == 10)

{

isNegative = true;

num = -num;

}

// Process individual digits

while (num != 0)

{

int rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

num = num/base;

}

// If number is negative, append '-'

if (isNegative)

str[i++] = '-';

str[i] = '\0'; // Append string terminator

// Reverse the string

reverse(str, i);

return str;

}

unsigned int get\_host\_ipn(const char\* name)

{

struct addrinfo\* addr = 0;

unsigned int ip4addr = 0;

// Функция возвращает все адреса указанного хоста

// в виде динамического однонаправленного списка

if (0 == getaddrinfo(name, 0, 0, &addr))

{

struct addrinfo\* cur = addr;

while (cur)

{

// Интересует только IPv4 адрес, если их несколько - то первый

if (cur->ai\_family == AF\_INET)

{

ip4addr = ((struct sockaddr\_in\*) cur->ai\_addr)->sin\_addr.s\_addr;

break;

}

cur = cur->ai\_next;

}

freeaddrinfo(addr);

}

return ip4addr;

}

int main(int argc ,char \*argv[])

{

if(argc!=3)

{

printf("Wrong");

return 0;

}

const char\* ip=strtok(argv[1],":");

int port=atoi(strtok(NULL," "));

struct my\_time data\_and\_message[N];

parsing\_file(argv[2],data\_and\_message);

struct sockaddr\_in addr;

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

if(s<0)

return sock\_err("socket",s);

memset(&addr,0,sizeof(addr));

addr.sin\_family=AF\_INET;

addr.sin\_port=htons(port);

addr.sin\_addr.s\_addr=get\_host\_ipn(ip);

for(int i=0;i<10;i++)

{

if(connect(s,(struct sockaddr\*)&addr,sizeof(addr))!=0)

{

usleep(100000);

}

if(i==9 && connect(s,(struct sockaddr\*)&addr,sizeof(addr)!=0))

{

close(s);

return sock\_err("connect",s);

}

else

{

char put='p';

int res=send(s,&put,1,0);

put='u';

res=send(s,&put,1,0);

put='t';

res=send(s,&put,1,0);

break;

}

}

send\_messages(data\_and\_message,s);

close(s);

return 0;

}

void parsing\_file(char\* name\_of\_file,struct my\_time\* data\_and\_message)

{

int i=0;

char datas[512]="\0";

char message[50000]="\0";

char\* data1="\0";

char\* data2="\0";

char\* data3="\0";

FILE\* f;

f=fopen(name\_of\_file,"r");

for(number\_of\_message=0;number\_of\_message<N;number\_of\_message++)

{

while(!(feof(f)))

{

i=0;

char c='1';

while(!isalpha(c))

{

c=getc(f);

datas[i]=c;

i++;

}

datas[strlen(datas)-1]='\0';

i=0;

fseek(f,-sizeof(char),SEEK\_CUR);

//c=getc(f);

data1=strtok(datas," ");

data2=strtok(NULL," ");

data3=strtok(NULL," ");

data\_and\_message[number\_of\_message].tm\_myday1=atoi(strtok(data1,"."));

data\_and\_message[number\_of\_message].tm\_mymonth1=atoi(strtok(NULL,"."));

data\_and\_message[number\_of\_message].tm\_myyear1=atoi(strtok(NULL,"."));

data\_and\_message[number\_of\_message].tm\_myday2=atoi(strtok(data2,"."));

data\_and\_message[number\_of\_message].tm\_mymonth2=atoi(strtok(NULL,"."));

data\_and\_message[number\_of\_message].tm\_myyear2=atoi(strtok(NULL,"."));

data\_and\_message[number\_of\_message].tm\_myhour1=atoi(strtok(data3,":"));

data\_and\_message[number\_of\_message].tm\_myminutes1=atoi(strtok(NULL,":"));

data\_and\_message[number\_of\_message].tm\_mysec1=atoi(strtok(NULL,":"));

memset(datas,0,strlen(datas));

memset(data1,0,strlen(data1));

memset(data2,0,strlen(data2));

memset(data3,0,strlen(data3));

int flag=0;

i=0;

while(message[i]=getc(f))

{

if(isdigit(message[i]))

{

flag++;

i++;

message[i]=getc(f);

}

if(isdigit(message[i]))

{

flag++;

i++;

message[i]=getc(f);

}

if(message[i]=='.')

{

flag++;

i++;

// message[i]=getc(f);

break;

}

if (feof(f))

break;

flag=0;

i++;

}

data\_and\_message[number\_of\_message].message=(char\*)calloc(strlen(message)+1,sizeof(char));

if(flag==3)

{

for(int i=strlen(message);i>0;i--)

{

if(message[i]=='\n' && message[i-1]=='\n')

{

memset(message+i-1,0,50000-i-1);

break;

}

if(message[i]=='\n')

{

memset(message+i,0,50000-i);

break;

}

}

fseek(f,-sizeof(char),SEEK\_CUR);

fseek(f,-sizeof(char),SEEK\_CUR);

fseek(f,-sizeof(char),SEEK\_CUR);

memcpy(data\_and\_message[number\_of\_message].message,message,strlen(message)+1);

memset(message,0,50000);

number\_of\_message++;

continue;

}

if(flag==2)

{

if(message[i]=='\377' || -1 ==int(message[i]) )

message[i]='\0';

memcpy(data\_and\_message[number\_of\_message].message,message,strlen(message)+1);

memset(message,0,50000);

number\_of\_message++;

continue;

}

if(flag==1)

{

//fprintf(stdout,"%s",message);

if(message[i]=='\377' || -1 ==int(message[i]) )

message[i]='\0';

memcpy(data\_and\_message[number\_of\_message].message,message,strlen(message)+1);

memset(message,0,50000);

number\_of\_message++;

continue;

}

if(flag==0)

{

message[strlen(message)-1]='\0';

memcpy(data\_and\_message[number\_of\_message].message,message,strlen(message)+1);

memset(message,0,50000);

}

number\_of\_message++;

memset(message,0,50000);

}

break;

}

}

void send\_messages(struct my\_time\* data\_and\_message, int s)

{

int number\_of\_message2=0;

int number\_of\_message2\_to\_send=0;

char the\_number\_to\_send;

char nmsg[5]="\0";

char one\_byte[2]="\0";

char ok[3]="\0";

for(number\_of\_message2=0;number\_of\_message2<number\_of\_message;number\_of\_message2++)

{

number\_of\_message2\_to\_send=htonl(number\_of\_message2);

memcpy(nmsg,&number\_of\_message2\_to\_send,4);

send(s,nmsg,4,MSG\_NOSIGNAL); // we are sinding the number of message

// now we are seinding the chislo of month

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_myday1);

//one\_byte[1]="\0";

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_mymonth1);

//one\_byte[1]="\0";

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

unsigned short year=htons(data\_and\_message[number\_of\_message2].tm\_myyear1);

send(s,&year,2,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_myday2);

//one\_byte[1]="\0";

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_mymonth2);

//one\_byte[1]="\0";

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

year=htons(data\_and\_message[number\_of\_message2].tm\_myyear2);

send(s,&year,2,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_myhour1);

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_myminutes1);

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

one\_byte[0]=char(data\_and\_message[number\_of\_message2].tm\_mysec1);

send(s,&one\_byte[0],1,MSG\_NOSIGNAL);

send(s,data\_and\_message[number\_of\_message2].message,strlen(data\_and\_message[number\_of\_message2].message)+1,MSG\_NOSIGNAL);

recv(s,ok,2,MSG\_NOSIGNAL);

fprintf(stdout,"%s\n",ok);

}

}

TCP-сервер

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifdef \_WIN32

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

// Директива линковщику: использовать библиотеку сокетов

#pragma comment(lib, "ws2\_32.lib")

#else // LINUX

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <errno.h>

#endif

#ifdef \_WIN32

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#else // LINUX

#include <fcntl.h>

#endif

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define N 10

struct clients

{

int s;

unsigned int ip;

};

int set\_non\_block\_mode(int s)

{

#ifdef \_WIN32

unsigned long mode = 1;

return ioctlsocket(s, FIONBIO, &mode);

#else

int fl = fcntl(s, F\_GETFL, 0);

return fcntl(s, F\_SETFL, fl | O\_NONBLOCK);

#endif

}

int counter = 0;

int connectedSocketsPut[50] = { 0 };

short port;

FILE\* fp;

bool stop;

int init()

{

#ifdef \_WIN32

// Для Windows следует вызвать WSAStartup перед началом использования сокетов

WSADATA wsa\_data;

return (0 == WSAStartup(MAKEWORD(2, 2), &wsa\_data));

#else

return 1; // Для других ОС действий не требуется

#endif

}

void deinit()

{

#ifdef \_WIN32

// Для Windows следует вызвать WSACleanup в конце работы

WSACleanup();

#else

// Для других ОС действий не требуется

#endif

}

int sock\_err(const char\* function, int s)

{

int err;

#ifdef \_WIN32

err = WSAGetLastError();

#else

err = errno;

#endif

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

#ifdef \_WIN32

closesocket(s);

#else

close(s);

#endif

}

bool is\_check = true;

void resv\_msg(int, struct clients\*);

int main(int argc,char\* argv[])

{

init();

port = atoi(argv[1]);

fp = fopen("msg.txt", "w+");

struct sockaddr\_in addr;

int ls;

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

if (ls < 0)

return sock\_err("soket", ls);

set\_non\_block\_mode(ls);

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(ls, (struct sockaddr\*) &addr, sizeof(addr)) < 0)

return sock\_err("bind", ls);

if (listen(ls, 100) < 0)

return sock\_err("listen", ls);

struct clients cs[N];

fd\_set rfd;

int nfds = ls;

int i;

struct timeval tv = { 1, 0 };

while (1)

{

FD\_ZERO(&rfd);

FD\_SET(ls, &rfd);

for (i = 0; i < counter; i++)

{

FD\_SET(cs[i].s, &rfd);

if (nfds < cs[i].s)

nfds = cs[i].s;

}

if (select(nfds + 1, &rfd, 0, 0, &tv) > 0)

{

// Есть события

if (FD\_ISSET(ls, &rfd))

{

// Есть события на прослушивающем сокете, можно вызвать accept,принять

// подключение и добавить сокет подключившегося клиента в массив cs

int addrlen = sizeof(struct sockaddr\_in);

cs[counter].s = accept(ls, (struct sockaddr\*) &addr, &addrlen);

if (cs[counter].s < 0)

{

return sock\_err("accept", ls);

}

set\_non\_block\_mode(cs[counter].s);

cs[counter].ip = ntohl(addr.sin\_addr.s\_addr);

fprintf(stdout, "Peer connected : %d.%d.%d.%d:%i\n", (cs[counter].ip >> 24) & 0xFF, (cs[counter].ip >> 16) & 0xFF, (cs[counter].ip >> 8) & 0xFF, (cs[counter].ip) & 0xFF, port);

counter++;

}

for (i = 0; i < counter; i++)

{

if (FD\_ISSET(cs[i].s, &rfd))

{

// Сокет cs[i] доступен для чтения. Функция recv вернет данные,recvfrom - дейтаграмму

resv\_msg(i,cs);

if (stop)

{

printf("'stop' arrived.");

fclose(fp);

for (int i = 0; i < counter; i++)

{

closesocket(cs[i].s);

goto metka;

}

}

}

}

}

else

{

// Произошел таймаут или ошибка

}

}

metka:

closesocket(ls);

return 0;

}

void resv\_msg(int num, struct clients \*cs)

{

Sleep(50);

char get\_put[4] = "\0";

int rec = 0;

if (connectedSocketsPut[num] != 1)

{

rec=recv(cs[num].s, get\_put, 3, 0);

connectedSocketsPut[num] = 1;

}

/\*char nmsg[4] = "\0";

recv(cs[num].s, nmsg, 3, 0);\*/

char get\_num[5] = "\0";

recv(cs[num].s, get\_num, 4, 0);

fprintf(fp, "%u.%u.%u.%u:%u ", (cs[num].ip >> 24) & 0xFF, (cs[num].ip >> 16) & 0xFF, (cs[num].ip >> 8) & 0xFF, (cs[num].ip) & 0xFF, port);

char day;

char month;

unsigned short year;

char year\_buf[2] = "\0";

for (int i = 0; i < 2; i++)

{

recv(cs[num].s, &day, 1, 0); //

if ((int)day < 10) fprintf(fp, "0");

fprintf(fp, "%i.", (int)day);

recv(cs[num].s, &month, 1, 0); //

if ((int)month < 10) fprintf(fp, "0");

fprintf(fp, "%i.", (int)month);

recv(cs[num].s, year\_buf, 2, 0); //

memcpy(&year, year\_buf, 2);

fprintf(fp, "%u ", htons(year));

}

char dma;

recv(cs[num].s, &dma, 1, 0);

if ((int)dma < 10) fprintf(fp, "0");

fprintf(fp, "%i:", (int)dma);

recv(cs[num].s, &dma, 1, 0);

if ((int)dma < 10) fprintf(fp, "0");

fprintf(fp, "%i:", (int)dma);

recv(cs[num].s, &dma, 1, 0);

if ((int)dma<10) fprintf(fp, "0");

fprintf(fp, "%i ", (int)dma);

char message[100000] = "\0";

memset(message, 0, 100000);

while ((rec = recv(cs[num].s, message, 100000, 0)) > 0);

//fputs(message, fp);

fprintf(fp, "%s", message);

if (strcmp(message, "stop"))

{

fputs("\n", fp);

}

//recv(cs[num].s, message, 35000, 0);

send(cs[num].s, "ok", 2, 0);

if (!strcmp(message, "stop"))

{

stop = 1;

}

}

UDP-клиент

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifdef \_WIN32

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

// Директива линковщику: использовать библиотеку сокетов

#pragma comment(lib, "ws2\_32.lib")

#else // LINUX

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <sys/select.h>

#include <netdb.h>

#include <errno.h>

#endif

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

unsigned int number\_of\_message;

char\* datagrams[20];

#define N 20000

int already\_recieved = 0;

unsigned int datagram[20];

FILE\* f=NULL;

void parsing\_file(char\* name\_of\_file, struct my\_time \*data\_and\_message);

int send\_your\_mess(int s, struct sockaddr\_in \*addr, struct my\_time\* data\_and\_message);

unsigned int recv\_response(int s, struct sockaddr\_in \*addr, struct my\_time\* data\_and\_message);

void resend\_msg(int s, struct sockaddr\_in \*addr, struct my\_time\* data\_and\_message);

struct my\_time

{

unsigned int inx;

unsigned int tm\_myday1;

unsigned int tm\_mymonth1;

unsigned short tm\_myyear1;

unsigned int tm\_myhour1;

unsigned int tm\_myminutes1;

unsigned int tm\_mysec1;

unsigned int tm\_myday2;

unsigned int tm\_mymonth2;

unsigned short tm\_myyear2;

char\* message;

};

int init()

{

#ifdef \_WIN32

// Для Windows следует вызвать WSAStartup перед началом использования сокетов

WSADATA wsa\_data;

return (0 == WSAStartup(MAKEWORD(2, 2), &wsa\_data));

#else

return 1; // Для других ОС действий не требуется

#endif

}

void deinit()

{

#ifdef \_WIN32

// Для Windows следует вызвать WSACleanup в конце работы

WSACleanup();

#else

// Для других ОС действий не требуется

#endif

}

int sock\_err(const char\* function, int s)

{

int err;

#ifdef \_WIN32

err = WSAGetLastError();

#else

err = errno;

#endif

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

#ifdef \_WIN32

closesocket(s);

#else

close(s);

#endif

}

int main(int argc,char\* argv[])

{

init();

int s;

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

if (s < 0)

return sock\_err("socket", s);

short port;

char \*ip=NULL;

struct sockaddr\_in addr;

//if (argc == 3)

//{

strtok(argv[1], ":");

ip = argv[1];

port = (short int)atoi(strtok(NULL, "\0"));

f = fopen(argv[2], "r");

//}

if (f == NULL)

{

printf("Error: cannot open file!\n");

return 0;

}

// Заполнение структуры с адресом удаленного узла

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port); // Порт DNS - 53

addr.sin\_addr.s\_addr = inet\_addr(ip);

printf(" Server address: %u.%u.%u.%u:%u\n",

(ntohl(inet\_addr(ip)) >> 24) & 0xFF, (ntohl(inet\_addr(ip)) >> 16) & 0xFF, (ntohl(inet\_addr(ip)) >> 8) & 0xFF, (ntohl(inet\_addr(ip))) & 0xFF, htons(port));

//send\_your\_mess(s, &addr);

struct my\_time data\_and\_message[N];

parsing\_file(argv[2], data\_and\_message);

send\_your\_mess(s, &addr, data\_and\_message);

while (already\_recieved <= 20 && already\_recieved!=number\_of\_message )

{

recv\_response(s, &addr, data\_and\_message);

//resend\_msg(s, &addr, data\_and\_message);

}

deinit();

return 0;

}

void parsing\_file(char\* name\_of\_file, struct my\_time\* data\_and\_message)

{

int i = 0;

char datas[512] = "\0";

char message[50000] = "\0";

char\* data1 = NULL;

char\* data2 = NULL;

char\* data3 = NULL;

int trh\_space = 0;

FILE\* f;

f = fopen(name\_of\_file, "r");

for (number\_of\_message = 0; number\_of\_message < 20 && !(feof(f)); number\_of\_message++)

{

//while (!(feof(f)))

//{

i = 0;

char c = '1';

while (!isalpha(c))

{

if (trh\_space == 3)

break;

c = getc(f);

if (c == ' ')

trh\_space++;

datas[i] = c;

i++;

}

trh\_space = 0;

datas[strlen(datas) - 1] = '\0';

i = 0;

//fseek(f, -1, SEEK\_CUR);

//c=getc(f);

data1 = strtok(datas, " ");

data2 = strtok(NULL, " ");

data3 = strtok(NULL, " ");

data\_and\_message[number\_of\_message].tm\_myday1 = atoi(strtok(data1, "."));

data\_and\_message[number\_of\_message].tm\_mymonth1 = atoi(strtok(NULL, "."));

data\_and\_message[number\_of\_message].tm\_myyear1 = atoi(strtok(NULL, "."));

data\_and\_message[number\_of\_message].tm\_myday2 = atoi(strtok(data2, "."));

data\_and\_message[number\_of\_message].tm\_mymonth2 = atoi(strtok(NULL, "."));

data\_and\_message[number\_of\_message].tm\_myyear2 = atoi(strtok(NULL, "."));

data\_and\_message[number\_of\_message].tm\_myhour1 = atoi(strtok(data3, ":"));

data\_and\_message[number\_of\_message].tm\_myminutes1 = atoi(strtok(NULL, ":"));

data\_and\_message[number\_of\_message].tm\_mysec1 = atoi(strtok(NULL, ":"));

memset(datas, 0, strlen(datas));

memset(data1, 0, strlen(data1));

memset(data2, 0, strlen(data2));

memset(data3, 0, strlen(data3));

fgets(message, 512, f);

if (!strcmp(message, "\n"))

continue;

if (!strcmp(message, "stop"))

{

message[strlen(message)] = '\0';

data\_and\_message[number\_of\_message].message = (char\*)calloc(strlen(message) + 1, sizeof(char));

memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message));

}

else

{

char\* mes1;

mes1 = strtok(message, "\n");

data\_and\_message[number\_of\_message].message = (char\*)calloc(strlen(message) + 1, sizeof(char));

memcpy(data\_and\_message[number\_of\_message].message, mes1, strlen(mes1));

}

//data\_and\_message[number\_of\_message].message = (char\*)calloc(strlen(message) + 1, sizeof(char));

//memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message));

//}

}

/\* c = getc(f);

int flag = 0;

i = 0;

while (message[i] = getc(f))

{

if (isdigit(message[i]))

{

flag++;

i++;

message[i] = getc(f);

}

if (isdigit(message[i]))

{

flag++;

i++;

message[i] = getc(f);

}

if (message[i] == '.')

{

flag++;

i++;

// message[i]=getc(f);

break;

}

if (feof(f))

break;

flag = 0;

i++;

}

data\_and\_message[number\_of\_message].message = (char\*)calloc(strlen(message) + 1, sizeof(char));

if (flag == 3)

{

for (int i = strlen(message); i>0; i--)

{

if (message[i] == '\n' && message[i - 1] == '\n')

{

memset(message + i - 1, 0, 50000 - i - 1);

break;

}

if (message[i] == '\n')

{

memset(message + i, 0, 50000 - i);

break;

}

}

fseek(f, -(int)sizeof(char), SEEK\_CUR);

fseek(f, -(int)sizeof(char), SEEK\_CUR);

fseek(f, -(int)sizeof(char), SEEK\_CUR);

memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message) + 1);

memset(message, 0, 50000);

number\_of\_message++;

continue;

}

if (flag == 2)

{

if (message[i] == '\377' || -1 == int(message[i]))

message[i] = '\0';

memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message) + 1);

memset(message, 0, 50000);

number\_of\_message++;

continue;

}

if (flag == 1)

{

//fprintf(stdout,"%s",message);

if (message[i] == '\377' || -1 == int(message[i]))

message[i] = '\0';

memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message) + 1);

memset(message, 0, 50000);

number\_of\_message++;

continue;

}

if (flag == 0)

{

message[strlen(message) - 1] = '\0';

memcpy(data\_and\_message[number\_of\_message].message, message, strlen(message) + 1);

memset(message, 0, 50000);

}

number\_of\_message++;

memset(message, 0, 50000);

}

break;

}

\*/

}

int send\_your\_mess(int s, struct sockaddr\_in \*addr, struct my\_time\* data\_and\_message)

{

//char \*str = (char\*)calloc(N, sizeof(char));

char str[50000] = "\0";

int size = 0;

int i;

int NETWORK\_num\_to\_send = 0;

int len = 0;

for (i = 0; i < number\_of\_message; i++)

{

data\_and\_message[i].inx = i;

NETWORK\_num\_to\_send = htonl(i);

memcpy(str, &NETWORK\_num\_to\_send, sizeof(int));

len = 4;

strncat(str+sizeof(NETWORK\_num\_to\_send), (const char\*)&data\_and\_message[i].tm\_myday1, 1);

len++;

strncat(str+ 5, (const char\*)&data\_and\_message[i].tm\_mymonth1, 1);

len++;

unsigned short year = htons(data\_and\_message[i].tm\_myyear1);

len += 2;

char god1[3] = "\0";

memcpy(god1, &year, 2);

strncat(str+ 6, god1, 2);

strncat(str+ 8, (const char\*)&data\_and\_message[i].tm\_myday2, 1);

len++;

strncat(str+ 9, (const char\*)&data\_and\_message[i].tm\_mymonth2, 1);

len++;

year = htons(data\_and\_message[i].tm\_myyear2);

memset(god1, 0, 3);

memcpy(god1, &year, 2);

strncat(str+ 10, god1, 2);

len += 2;

strncat(str+ 12, (const char\*)&data\_and\_message[i].tm\_myhour1, 1);

len++;

strncat(str+ 13, (const char\*)&data\_and\_message[i].tm\_myminutes1, 1);

len++;

strncat(str+ 14, (const char\*)&data\_and\_message[i].tm\_mysec1, 1);

len++;

int oldlen = strlen(data\_and\_message[i].message);

len = len + oldlen;

strncat(str + 15, data\_and\_message[i].message, strlen(data\_and\_message[i].message));

int res = sendto(s, str, len + 1, 0, (struct sockaddr\*)addr, sizeof(struct sockaddr\_in));

memset(str, 0, len + 1);

}

return 0;

}

void resend\_msg(int s, sockaddr\_in \*addr,struct my\_time\* data\_and\_message)

{

char str[50000] = "\0";

int size = 0;

int i;

int NETWORK\_num\_to\_send = 0;

int len = 0;

for (i = 0; i < number\_of\_message && data\_and\_message[i].tm\_myday1!=0; i++)

{

//data\_and\_message[i].inx = i;

NETWORK\_num\_to\_send = htonl(data\_and\_message[i].inx);

memcpy(str, &NETWORK\_num\_to\_send, sizeof(int));

len = 4;

strncat(str + sizeof(NETWORK\_num\_to\_send), (const char\*)&data\_and\_message[i].tm\_myday1, 1);

len++;

strncat(str + 5, (const char\*)&data\_and\_message[i].tm\_mymonth1, 1);

len++;

unsigned short year = htons(data\_and\_message[i].tm\_myyear1);

len += 2;

char god1[3] = "\0";

memcpy(god1, &year, 2);

strncat(str + 6, god1, 2);

strncat(str + 8, (const char\*)&data\_and\_message[i].tm\_myday2, 1);

len++;

strncat(str + 9, (const char\*)&data\_and\_message[i].tm\_mymonth2, 1);

len++;

year = htons(data\_and\_message[i].tm\_myyear2);

memset(god1, 0, 3);

memcpy(god1, &year, 2);

strncat(str + 10, god1, 2);

len += 2;

strncat(str + 12, (const char\*)&data\_and\_message[i].tm\_myhour1, 1);

len++;

strncat(str + 13, (const char\*)&data\_and\_message[i].tm\_myminutes1, 1);

len++;

strncat(str + 14, (const char\*)&data\_and\_message[i].tm\_mysec1, 1);

len++;

int oldlen = strlen(data\_and\_message[i].message);

len = len + oldlen;

strncat(str + 15, data\_and\_message[i].message, strlen(data\_and\_message[i].message));

int res = sendto(s, str, len + 1, 0, (struct sockaddr\*)addr, sizeof(struct sockaddr\_in));

memset(str, 0, len + 1);

}

}

unsigned int recv\_response(int s, sockaddr\_in \*send\_addr, struct my\_time\* data\_and\_message)

{

struct timeval tv = { 0, 100 \* 1000 }; // 100 msec

fd\_set fds;

FD\_ZERO(&fds);

FD\_SET(s, &fds);

int received = 0;

struct sockaddr\_in addr;

int addrlen = sizeof(addr);

int res = 0;

do

{

FD\_ZERO(&fds);

FD\_SET(s, &fds);

// Проверка - если в сокете входящие дейтаграммы

// (ожидание в течение tv)

res = select(s + 1, &fds, 0, 0, &tv);

if (res > 0)

{ // Данные есть, считывание их

received = recvfrom(s, (char\*)datagram, sizeof(datagram), 0, (struct sockaddr \*)&addr, &addrlen);

if (received <= 0)

{

// Ошибка считывания полученной дейтаграммы

sock\_err("recvfrom", s);

return 0;

}

received /= sizeof(int);

// Удаляем информацию о датаграммах,полученных сервером

for (int i = 0; i < received; i++)

{

datagram[i] = ntohl(datagram[i]);

if (datagram[i] <= 20 && data\_and\_message[datagram[i]].tm\_myday1 != 0)

{

already\_recieved++;

data\_and\_message[datagram[i]].tm\_myday1 = 0;// зануляются те сообщения,что были отправлены

}

}

// Повторно отправлеям неполученные датаграммы

resend\_msg(s, &addr, data\_and\_message);

}

} while (res>0);

return 0;

}

UDP-сервер

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/time.h>

#include <poll.h>

#include <netdb.h>

#include <errno.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <ifaddrs.h>

#include <stdlib.h>

#include <fcntl.h>

#include<stdlib.h>

#include <iostream>

using namespace std;

FILE \*f;

bool stop;

int cscnt;

unsigned int \*msg\_cnt;

//#define N 5535

#define N 20000

struct client

{

unsigned int ip;

unsigned short int port;

int idx[50];

};

client \*clients;

int set\_non\_block\_mode(int ls)

{

int fl = fcntl(ls, F\_GETFL, 0);

return fcntl(ls, F\_SETFL, fl | O\_NONBLOCK);

}

int sock\_err(const char \*function, int ls)

{

int err;

err = errno;

fprintf(stderr, "%ls: socket error: %d\n", function, err);

return -1;

}

int recv\_msg(int,struct sockaddr\_in);

/\* A utility function to reverse a string \*/

void reverse(char str[], int length)

{

int start = 0;

int end = length -1;

while (start < end)

{

swap(\*(str+start), \*(str+end));

start++;

end--;

}

}

// Implementation of itoa()

char\* itoa(int num, char\* str, int base)

{

int i = 0;

bool isNegative = false;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

if (num == 0)

{

str[i++] = '0';

str[i] = '\0';

return str;

}

// In standard itoa(), negative numbers are handled only with

// base 10. Otherwise numbers are considered unsigned.

if (num < 0 && base == 10)

{

isNegative = true;

num = -num;

}

// Process individual digits

while (num != 0)

{

int rem = num % base;

str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

num = num/base;

}

// If number is negative, append '-'

if (isNegative)

str[i++] = '-';

str[i] = '\0'; // Append string terminator

// Reverse the string

reverse(str, i);

return str;

}

int fill\_in(struct sockaddr\_in \*addr)

{

unsigned int ip = ntohl(addr->sin\_addr.s\_addr);

unsigned short int port = ntohs(addr->sin\_port);

for (int i = 0; i < cscnt; i++)

{

if (clients[i].ip == ip && clients[i].port == port)

return i;

}

cscnt++;

clients = (client \*)realloc(clients, (cscnt+1) \* sizeof(client));

clients[cscnt - 1].ip = ip;

clients[cscnt - 1].port = port;

for (int i = 0; i < 50; i++)

clients[cscnt - 1].idx[i] = -1;

printf("New client detected: %u.%u.%u.%u:%u\n",

(ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, (ip)&0xFF, port);

return cscnt - 1;

}

bool add\_idx(int num, int idx)

{

int j = -1;

for (int i = 0; i < 50; i++)

{

if (clients[num].idx[i] == idx)

{

return false;

}

if (clients[num].idx[i] == -1 && j == -1)

j = i;

}

if (j >= 0 && j < 50)

clients[num].idx[j] = idx;

return true;

}

int send\_notice(int num, int s, sockaddr\_in \*addr)

{

char str[80] = {'\0'};

char \*ptr = str;

// формирвоание ответа

for (int i = 0; i < 20; i++)

{

if (clients[num].idx[i] != -1)

{

int idx = htonl(clients[num].idx[i]);

//int id = clients[num].idx[i];

memcpy(ptr, &idx, sizeof(int));

ptr += 4;

}

}

// Отправка ответа

int res = sendto(s, str, ptr - str,

0, (struct sockaddr \*)addr, sizeof(struct sockaddr\_in));

if (res <= 0)

sock\_err("sendto", s);

}

int main(int argc, char\* argv[])

{

short int port1, port2;

if (argc == 2)

{

port1 = (short int)atoi(argv[1]);

port2 = port1;

}

else if (argc == 3)

{

port1 = (short int)atoi(argv[1]);

port2 = (short int)atoi(argv[2]);

if(port2<port1){

printf("No port to listen\n");

return 0;

}

}

else

{

port1 = 8700;

port2 = 8700;

}

const int port\_num = port2 - port1 + 1;

int \*ls = (int \*)calloc(port\_num, sizeof(int)); // Создание UDP-сокета

msg\_cnt = (unsigned int \*)calloc(port\_num, sizeof(int));

clients = (client \*)calloc(1, sizeof(client)); // Массив инфорации

struct sockaddr\_in addr[N];

int flags = MSG\_NOSIGNAL;

int nfds = 0;

f = fopen("msg.txt", "a+");

struct pollfd \*pfd;

pfd=(pollfd\*)malloc(sizeof(pollfd)\*port\_num);

// Привязка сокетов

printf("Listening on:");

for (int i = 0; i < port\_num; i++)

{

ls[i] = socket(AF\_INET, SOCK\_DGRAM, 0);

if (ls < 0)

return sock\_err("socket", ls[i]);

set\_non\_block\_mode(ls[i]);

// Заполнение структуры с адресом прослушивания узла

memset(&addr, 0, sizeof(addr));

addr[i].sin\_family = AF\_INET;

addr[i].sin\_port = htons(i + port1); // Будет прослушиваться порт 8000

addr[i].sin\_addr.s\_addr = htonl(INADDR\_ANY);

// Связь адреса и сокета, чтобы он мог принимать входящие дейтаграммы

if (bind(ls[i], (struct sockaddr \*)&addr[i], sizeof(addr[i])) < 0)

return sock\_err("bind", ls[i]);

pfd[i].fd = ls[i];

pfd[i].events = POLLIN | POLLOUT ;

}

while (1)

{

int ev\_cnt = poll(pfd, port\_num, 1000); // Waiting events for 1 sec

if (ev\_cnt > 0)

{

for (int i = 0; i < port\_num; i++)

{

if (pfd[i].revents & POLLHUP)

{

// Client disconnected, closing socket

for(int j=0;j<port\_num;j++)

{

close(ls[j]);

return 0;

}

}

if (pfd[i].revents & POLLERR) // Error, closing socket

{

for(int j=0;j<port\_num;j++)

{

close(ls[j]);

return 0;

}

}

if (pfd[i].revents & POLLIN) // Sock\_arr[i] avaible for reading, call recv

{

recv\_msg(ls[i],addr[i]);

if (stop)

{

printf("'stop' message arrived. Terminating...\n");

//fclose(f);

// Закрытие сокета

for (int i = 0; i < port\_num; i++)

close(ls[i]);

free(ls);

free(clients);

free(msg\_cnt);

free(pfd);

fclose(f);

return 0;

}

}

}

}

}

}

int recv\_msg(int cs,struct sockaddr\_in addr)

{

int flags = MSG\_NOSIGNAL;

//struct sockaddr\_in addr;

socklen\_t addrlen = sizeof(addr);

char buffer[N] = {'\0'};

//char str[N] = {'\0'};

char \*buf = buffer;

int rcv = recvfrom(cs, (char \*)buffer, sizeof(buffer) - 1, flags, (struct sockaddr \*)&addr, &addrlen);

if (rcv < 0)

return sock\_err("recv", cs);

if (rcv == 0)

return 0;

int num= fill\_in(&addr);// number of client has right come

int idx;

memcpy(&idx, buf, 4);

if (!add\_idx(num, ntohl(idx)))

{

return 0;

}

buf += 4;

unsigned int ip = clients[num].ip;

fprintf(f, "%u.%u.%u.%u:%u ",

(ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, (ip)&0xFF, clients[num].port);

char day;

char month;

char year\_buf[2]="\0";

unsigned short year=0;

for(int i=0;i<2;i++)

{

memcpy(&day,buf,1);// got the first day

buf++;

if ((int)day < 10) fprintf(f, "0");

fprintf(f, "%i.", (int)day);

memcpy(&month,buf,1);// got the first day

buf++;

if ((int)month < 10) fprintf(f, "0");

fprintf(f, "%i.", (int)month);

///////

memcpy(year\_buf,buf,2);

buf=buf+2;

memcpy(&year, year\_buf, 2);

fprintf(f, "%u ", htons(year));

}

char dma;

memcpy(&dma,buf,1);// got the first day

buf++;

if ((int)dma < 10) fprintf(f, "0");

fprintf(f, "%i:", (int)dma);

memcpy(&dma,buf,1);// got the first day

buf++;

if ((int)dma < 10) fprintf(f, "0");

fprintf(f, "%i:", (int)dma);

memcpy(&dma,buf,1);// got the first day

buf++;

if ((int)dma<10) fprintf(f, "0");

fprintf(f, "%i ", (int)dma);

char message[N]="\0";

memcpy(message,buf,strlen(buf));

fprintf(f,"%s",message);

if(!strcmp(message,"stop"))

{

stop=1;

}

if(strcmp(message,"stop"))

{

fputs("\n", f);

}

msg\_cnt[num]++;

send\_notice(num, cs, &addr);

memset(message,0,N);

buf=NULL;

memset(buffer,0,N);

return 0;

}

Выводы

В результате выполнения данной лабораторной работы был изучен принцип сетевых взаимодействий, некоторые алгоритмы параллельного обслуживания, а также получен практический опыт в написании программ типа клиент и сервер.