Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Системного программирования

ПРАКТИЧЕСКАЯ РАБОТА
по дисциплине
РАСПРЕДЕЛЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

Выполнил:
студент
Абакаров Г.Г.
группа БВТ1902

Проверил:
_____

Оценка_____

Дата _____

Москва 2022

ЦЕЛЬ КУРСОВОЙ РАБОТЫ

Целью данной курсовой работы является закрепление и углубление теоретических знаний в области современных операционных систем, приобретение практических навыков разработки утилит, используюя стандартные библиотеки c/c++.

ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

Разработать утилиту ping, реализуемое на основе технологии raw-socket, выполняющее проверки целостности и качества соединений в сетях на основе TCP/IP.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

Название сетевой модели TCP/IP произошло от сокращения двую английских выражении — Transmission Control Protocol и Internet Protocol. Протокол TCP/IP основан на OSI и имеет 4 уровня: Канальный уровень, Межсетевой уровень, Транспортный уровень, Прикладной уровень. В данной курсовой работе необходимо реализовать прикладной уровень протокола TCP/IP. Процессы, которые работают на прикладном уровне, взаимодействуют с транспортным уровнем протокола.
Процессы знают, на каком IP-адрес и порт адресованы данные.
Комбинация IP-адреса и порта называется сокетом, является файловым дескриптором, и используется для идентификации компьютера в сети.

Для работы с сокетами в C++ используется фукция *socket(int domain, int type, int protocol);*. Функция принимает 3 аргумента:
1. domain, указывающий семейство протоколов создаваемого сокета
- AF_INET для сетевого протокола IPv4
- AF_INET6 для IPv6
- AF_UNIX для локальных сокетов (используя файл)
2. type
- SOCK_STREAM (надёжная потокоориентированная служба (сервис) или потоковый сокет)
- SOCK_DGRAM (служба датаграмм или датаграммный сокет)
- SOCK_RAW (Сырой сокет — сырой протокол поверх сетевого уровня).
3. protocol

Протоколы обозначаются символьными константами с префиксом IPPROTO_* (например, IPPROTO_TCP или IPPROTO_UDP). Допускается значение protocol=0 (протокол не указан), в этом случае используется значение по умолчанию для данного вида соединений.

Чтобы связать сокет с конкретным адресом, используется функция *bind(int sockfd, const struct sockaddr * my_addr, socklen_t addrlen);*
Функция принимает три аргумента:

1. sockfd — дескриптор, представляющий сокет при привязке
2. serv_addr — указатель на структуру sockaddr, представляющую адрес, к которому привязываем.
3. addrlen — поле socklen_t, представляющее длину структуры sockaddr.

Чтобы подготовить сокет к принятию входящих сообщении, используется функция *listen(int sockfd, int backlog);*
Функция принимает 2 аргумента:
1. sockfd — дескриптор, представляющий сокет
2. backlog — целое число, которое означает число установленных соединении, которые могут быть обработаны в любой момент времени.

Для принятия запроса на установление соединения от удаленного хоста используется функция accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen); Принимает следующие аргументы:
1. sockfd — дескриптор слушающего сокета на принятие соединения.
2. cliaddr — указатель на структуру sockaddr, для принятия информации об адресе клиента.
3. addrlen — указатель на socklen_t, определяющее размер структуры, содержащей клиентский адрес и переданной в accept(). Когда accept() возвращает некоторое значение, socklen_t указывает сколько байт структуры cliaddr использовано в данный момент.

Чтобы установить соединение с сервером, используется функция connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
Некоторые типы сокетов работают без установления соединения, это в

основном касается UDP-сокетов. Для них соединение приобретает особое значение: цель по умолчанию для посылки и получения данных присваивается переданному адресу, позволяя использовать такие функции как send() и recv() на сокетах без установления соединения. Загруженный сервер может отвергнуть попытку соединения, поэтому в некоторых видах программ необходимо предусмотреть повторные попытки соединения.

Для передачи данных используется функция send (int s, const void *buf, size_t len, int flags); Функция принимает 4 аргумента:
1. s - дескриптор сокета
2. buf — указатель на буффер передаваемых данных
3. len — размер буффера
4. flags
- MSG_CONFIRM - Сообщить уровню связи, что процесс пересылки произошел
- MSG_DONTROUTE - Не использовать маршрутизацию для отправки пакета, а посылать его только на узлы локальной сети.
- MSG_DONTWAIT - Включить неблокирующий режим.
- MSG_EOR - Завершить запись (если поддерживается)
- MSG_MORE - Ожидать дополнительных данных для отправки.
- MSG_NOSIGNAL - Не генерировать сигнал SIGPIPE, если сторона потокоориентированного сокета закрыла соединение.
- MSG_OOB - Послать внепоточные данные, если сокет это поддерживает.

Для получения данных используется функция recv (int s, void *buf, size_t len, int flags);
1. s - дескриптор сокета
2. buf — указатель на буффер передаваемых данных
3. len — размер буффера
4. flags
 - MSG_ERRQUEUE - Указание этого флага позволяет получить из очереди ошибок сокета накопившиеся ошибки.

- MSG_DONTROUTE - Не использовать маршрутизацию для отправки пакета, а посылать его только на узлы локальной сети.
- MSG_DONTWAIT - Включить неблокирующий режим.
- MSG_WAITALL — Ожидать полной обратботки запроса.
- MSG_NOSIGNAL - Не генерировать сигнал SIGPIPE, если сторона потокоориентированного сокета закрыла соединение.
- MSG_OOB - Запросить внепоточные данные, если сокет это поддерживает.
- MSG_PEEK - Этот флаг заставляет выбрать данные из начала очереди приёма, но не удалять их оттуда.

Ping — утилита для проверки целостности и качества соединений в сетях на основе TCP/IP, а также обиходное наименование самого запроса.

Название происходит от английского названия звука импульса, издаваемого сонаром. Первоначально словом «ping» (по созвучию) именовали направленный акустический сигнал противолодочных гидролокаторов или «асдиков» (англ. ASDIC, аббревиатура от Allied Submarine Detection Investigation Committee).

Утилита отправляет запросы (ICMP Echo-Request) протокола ICMP указанному узлу сети и фиксирует поступающие ответы (ICMP Echo-Reply). Время между отправкой запроса и получением ответа (RTT, от англ. Round Trip Time) позволяет определять двусторонние задержки по маршруту и частоту потери пакетов, то есть косвенно определять загруженность на каналах передачи данных и промежуточных устройствах.

ICMP, или Internet Control Message Protocol — протокол межсетевых управляющих сообщений — сетевой протокол, входящий в стек протоколов TCP/IP. В основном ICMP используется для передачи сообщений об ошибках и других исключительных ситуациях, возникших при передаче данных, например, запрашиваемая услуга недоступна или хост, или маршрутизатор не отвечают.

ЛИСТИНГ ПРОГРАММЫ

```
// sockaddr_in, sockaddr,
#include <netinet/in.h>
// inet_aton, inet_ntoa
#include <arpa/inet.h>
// gethostbyname, hostent
#include <netdb.h>
// getpid
#include <unistd.h>
// timeval, gettimeofday, time
#include <sys/time.h>
#include <time.h>
// srand, rand
#include <stdlib.h>
// printf
#include <iostream>
// thread
#include <thread>
// mutex
#include <mutex>
// signal, kill
#include <signal.h>
// cout, endl
#include <iostream>
//
#include <string.h>
//
using namespace std;
namespace interrupt{
    const char * path;
    int * sent;
    int * received;
    void handler(int dummy){
        int snt = *sent;
        int rec = *received;
        cout<<endl
            <<"==="
```

```
            <<path
            <<" statistics==="
            <<endl
            <<snt
            <<" packets transmitted, "
            <<rec
            <<" packets received, "
            <<((snt!=0)?((snt-rec)/snt)*100.0:0.0)
            <<" percent packet lost"
            <<endl;
        exit(0);
    }
}

// заполнить адрес
bool dns_lookup(const char *addr_host, sockaddr_in* addr_con){
    //
    addr_con->sin_port = htons(8080);
    //
        struct hostent *host_entity;
    //
    if (inet_aton(addr_host, &addr_con->sin_addr) > 0){
        //
        addr_con->sin_family = AF_INET;
        return true;
    }
    //
        if ((host_entity = gethostbyname(addr_host)) == nullptr){
                // адрес не найден
                return false;
        }

        //
        addr_con->sin_family = host_entity->h_addrtype;
        addr_con->sin_addr.s_addr = *(long*)host_entity->h_addr;
    //
        return true;
```

```
}

#define MTU 1500
#define RECV_TIMEOUT_USEC 100000

struct icmp{
    uint8_t type;
    uint8_t code;
    uint16_t checksum;
    uint16_t ident;
    uint16_t seq;
};

struct ip
{
    uint8_t VIHL;
    uint8_t TOS;
    uint16_t TotLen;
    uint16_t id;
    uint16_t flagoff;
    uint8_t ttl;
    uint8_t protocol;
    uint16_t checksum;
    struct in_addr iaSrc;
    struct in_addr iaDst;
};

// функция вычисляет хеш сумму
uint16_t calculate_checksum(u_char* buffer, int bytes){
    uint32_t checksum = 0;
    //
    u_char* end = buffer + bytes;
    //
    if (bytes % 2 == 1) {
        end = buffer + bytes - 1;
        checksum += (*end) << 8;
    }
```

```
    //
    while (buffer < end) {
        checksum += buffer[0] << 8;
        checksum += buffer[1];
        buffer += 2;
    }
    //
    uint32_t carray = checksum >> 16;
    while (carray) {
        checksum = (checksum & 0xffff) + carray;
        carray = checksum >> 16;
    }
    //
    checksum = ~checksum;
    return checksum & 0xffff;
};

// функция генерирует случайный массив латинских букв
void random_chars(char * str, int size){
    //
    srand(time(nullptr));
    // продолжить пока size > 0
    while (size > 0){
        // присвоить рандомный номер
        *str = 'a' + (rand()%26);
        // увеличить указатель
        str++;
        size--;
    }
}

// функция отправляет пакет на сервер
int send_echo_request(int sock, struct sockaddr_in* addr, int ident,
    int seq, char * data, int data_len, timeval*start, timeval*stop){
    //
    auto ic = (icmp*) data;
    ic->checksum = 0;
```

```
    // заполнить icmp
    ic->type = 8;
    ic->code = 0;
    ic->ident = htons(ident);
    ic->seq = htons(seq);

    // вычислить хеш-сумму и записать
    ic->checksum = htons(calculate_checksum((u_char*)data, data_len));

    gettimeofday(start, nullptr);
    // оптравить
    int bytes = sendto(sock, data, data_len, 0,
        (struct sockaddr*)addr, sizeof(*addr));
    //
    gettimeofday(stop, nullptr);
    //
    if (bytes == -1) return -1;
    //
    return 0;
}

int recv_echo_reply(int sock, int ident, in_addr_t t, int seq, int *i,
        timeval*start, timeval*stop, char * buffer, int reply_size){
    struct sockaddr_in peer_addr;

    // receive another packet
    socklen_t addr_len = sizeof(peer_addr);
    //
    gettimeofday(start, nullptr);
    // bytes
    int bytes = recvfrom(sock, buffer, reply_size, 0,
        (struct sockaddr*)&peer_addr, &addr_len);
    //
    gettimeofday(stop, nullptr);
    //
    if (bytes == -1) {
        // normal return when timeout
```

```
        if (errno == EAGAIN || errno == EWOULDBLOCK) {
            return 0;
        }

        return -1;
    }

    // find icmp packet in ip packet
    auto ic = (struct icmp*)(buffer + 20);

    // проверить тип
    if (ic->type != 0 || ic->code != 0) {
        return 1;
    }

    // проверить идентификатор
    if (ntohs(ic->ident) != ident) {
        return 1;
    }

    // проверить seq
    if (seq == ic->seq){
        return 1;
    }

    // проверить адрес
    if (t != peer_addr.sin_addr.s_addr);
    //
    *i = bytes;

    return 0;
}

void dns_error(const char * name){
    cout<<"cannot find name "<< name <<endl;
}
```

```
void sock_error(){
    cout<<"cannot create raw socket"<<endl;
}

void dest_rec(){
    cout<<"destination address required"<<endl;
}

void time_error(timeval tv){
    cout<<"cannot set timeout of "<<
    ((tv.tv_sec*1000000)+tv.tv_usec)<<" ms"<<endl;
}

void send_error(){
    cout<<"send failure"<<endl;
}

void ttl_error(int ttl){
    cout<<"cannot set ttl of "<<ttl<<endl;
}

void recv_error(){
    cout<<"receive failure"<<endl;
}

//
void sleep(timeval&start, timeval&stop, timeval&max){
    //
    int i;
    //
    i = max.tv_sec - (stop.tv_sec - start.tv_sec);
    if (i > 0) sleep(i);
     //
    i = max.tv_usec - (stop.tv_usec - start.tv_usec);
    if (i > 0) usleep(i);
}
```

```
// pind
void ping(const char *name, int size = 44, int ttl = 65, int count = -
1,
          timeval tv = {0, 100000}, timeval step = {1, 0}){
    // создать буфер
    auto buf = new char[size];
    //
    int reply_size = size + 512;
    // создать буфер ответа
    auto reply = (ip*) new char[reply_size];
    // получить и проверить адрес
    struct sockaddr_in addr;
    if (!dns_lookup(name, &addr)) {dns_error(name); return;};
    // создать сокет для протокола icmp
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    // проверить сокет
    if (sock == -1) {sock_error(); return;};
    // присвоить значение ttl (time to live)
    if (setsockopt(sock,SOL_IP, IP_TTL, &ttl, sizeof(int))==-1){
        ttl_error(ttl); return;
    };
    // присвоить таймаут
    if (setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,&tv,sizeof(tv))==-1){
        time_error(tv); return;
    };

    timeval start1 = {0,0},
            start2 = {0,0},
            stop1  = {0,0},
            stop2  = {0,0};
    //

    int pid = getpid();

    bool run = true;
    ulong seq = 0;
    //
```

```
int i = 0, bytes = 0;
// пропустить ожидание
// create lock
auto lock = new mutex;
// run thread
new thread([](mutex*lock, timeval * vl)->void{
    l1:
    // sleep
    sleep(vl->tv_sec);
    usleep(vl->tv_usec);
    // unlock thread
    lock->unlock();
    goto l1;
}, lock, &step);
//
int sent = 0;
int received = 0;
//
interrupt::sent = &sent;
interrupt::received = &received;
interrupt::path = name;
//
signal(SIGINT, interrupt::handler);
//
l1:{
    //
    if (count == 0) return;
    else if (count > 0) count --;
    //
    if (i == 1) { goto l2; }
    // увеличить seq
    seq ++;
    //
    lock->lock();
    // создать случайную последовательность
    random_chars(buf, size);
    //
```

```
gettimeofday(&start1, nullptr);
// отправить пакет
if (send_echo_request(sock, &addr, pid,
                seq, buf, size, &start1, &stop1)==-1)
// вывести возможную ошибку
    {send_error(); goto l1; }
else{
    sent ++;
}
gettimeofday(&stop1, nullptr);
l2:

//
gettimeofday(&start2, nullptr);
// получить пакет
if ((i = recv_echo_reply(sock, pid, addr.sin_addr.s_addr, seq,
    &bytes, &start2, &stop2, (char*)reply, reply_size))==-1)
// вывести возможную ошибку
    {recv_error(); goto l1; }
else if (i == 0){
    received ++;
};
//
gettimeofday(&stop2, nullptr);

cout<<bytes
    <<" bytes from "
    <<inet_ntoa(reply->iaSrc)
    <<": icmp_seq="
    <<seq
    <<" ttl="
    <<(unsigned int)reply->ttl
    <<" time="
    <<((stop1.tv_sec-start1.tv_sec+
        stop2.tv_sec-start2.tv_sec)*1000000 +
       (stop1.tv_usec-start1.tv_usec+
        stop2.tv_usec-start2.tv_usec))
```

```
            <<" ms"<<endl;
      //
        goto l1;
    }

    return ;
}

void ms_to_timeval(int ms, timeval* tv){
    tv->tv_sec  = ms / 1000000;
    tv->tv_usec = ms % 1000000;
}

int main(int argc, char** argv){
    //
    int size = 44;
    int ttl = 65;
    int count = -1;
    //
    timeval tv = {0, 100000};
    //
    timeval step = {1, 0};
    //
    char * path = nullptr;
    //
    char * u;
    //
    char operation = 0;;
    //
    for (int i = 1; i < argc; i ++){
        //
        argv++;
        //
        u = *(argv);
        // проверить, начинается ли выражение с -
        // если да, то выполнить операцию
        if (u[0] == 0) continue;
```

```
        if (u[0] == '-'){
            // если длина строки 2, то проверить выражение
            if (strlen(u)==2){
                //
                operation = u[1];
                //
                if (operation == 'h'){
cout<<endl<<"Usage"<<endl<<"  ping [options] <destination>"
<<endl<<endl<<"Options:"
<<"  <destination>      dns name or ip address"<<endl
<<"  -c <count>         stop after <count> replies"<<endl
<<"  -l <size>          use <size> as number of data bytes to be sent"
<<endl<<"  -t <ttl>           define time to live"<<endl
<<"  -w <timeout>       time to wait for response"<<endl<<
"  -i <interval>      seconds between sending each
packet"<<endl<<endl;
                    exit(0);
                }
                else{

        argv++; i++;
        //
        if (i == argc) break;
        //
        if (operation != 0){
            switch (operation){
                case 't':
                ttl = atoi(*argv);
                break;
                case 'l':
                size = atoi(*argv);
                break;
                case 'i':
                ms_to_timeval(atoi(*argv), &step);
                break;
                case 'c':
                count = atoi(*argv);
```

```
                break;
                case 'w':
                ms_to_timeval(atoi(*argv), &tv);
                default:
                break;
            };
            operation = 0;
            continue;
        }
                }
            }//
            //
            continue;
        }
        //
        if (path == nullptr){
            path = u;
        }
    }
    //
    ping(path, size, ttl, count, tv, step);
    //
    kill(getpid(), SIGINT);
}
```