Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

по дисциплине «Операционные системы»

Выполнил
студент гр. 23508/4 Е.Г. Проценко

Проверил
преподаватель Е.Ю. Резединова

Санкт-Петербург
2016

1. Формулировка задания

Цель работы — изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

2. Теоретические сведения

Сетевой сокет — это одно из средств коммуникации процессов. Главное отличие сокетов от других средств межпроцессного взаимодействия — обменивающиеся информацией процессы могут быть удаленными, т.е. они не обязательно должны находится на одном компьютере.

UDP-клиенты

В отличие от TCP протокол UDP не поддерживает установку соединения, не гарантирует порядок доставки пакетов и доставку вообще. Протоколы передачи данных и файлов, основанные на UDP, должны предусматривать отправку подтверждений и повторных пакетов, чтобы обеспечить доставку данных.
Данные протокола UDP отправляются "дейтаграммами" — небольшими блоками данных. Отправленный блок будет доставлен на сервер полностью или не доставлен совсем. Возможна доставка повторных или ранее отправленных пакетов.
Отправка и прием данных осуществляются с помощью функций *sendto* и *recvfrom*:

```
int sendto(int sockfd, const void *buf, int len, int flags,
const struct sockaddr *dest_addr, int addrlen);
```

где *sockfd* — дескриптор сокета, *buf* — адрес буфера, содержащего передаваемые данные, *len* — размер передаваемых данных, *flags* — набор флагов: в Windows может быть передан 0, в Linux — рекомендуется применять *MSG_NOSIGNAL* (не прерывать выполнение программы в случае разрыва связи), *dest_addr* — адрес удаленной стороны, *addrlen* — размер данных, занимаемых *dest_addr*. Функция возвращает количество отправленных данных (совпадает с len), либо -1 в случае ошибки.

```
int recvfrom(int s, void *buf, int len, int flags,
struct sockaddr *from, int *fromlen);
```

где *sockfd* — дескриптор сокета, *buf* — адрес буфера, в который будут записаны принятые данные, *len* — размер буфера, *flags* — набор флагов: может быть передан 0. Функция возвращает количество принятых данных (размер принятой дейтаграммы), либо -1 — в случае ошибки. В переменную *from* сохраняется адрес удаленной стороны, приславшей дейтаграмму, в

переменную *fromlen* — размер сохраненных в *from* данных. При вызове функции *fromlen* должна содержать максимальный размер, который допустимо записывать по адресу *from*.

Следует отметить, что функция *recvfrom* не возвращает управление до тех пор, пока какая-либо дейтаграмма не будет получена от удаленной стороны, т.е. управление программой *блокируется* на все время работы функции. Такое поведение называется *блокирующим* режимом работы сокетов.

Если удаленная сторона не отправляет данные или они по каким-либо причинам не доставляются, то программа может навсегда "зависнуть" в функции ожидания очередной дейтаграммы. Чтобы исключить такое поведение следует проверить наличие данных в буфере приема UDP-сокета и, убедившись, что дейтаграмма присутствует, вызывать *recvfrom*.

UDP-серверы

Реализация UDP-сервера практически не отличается от реализации UDP-клиента. Исключением является необходимость "привязки" сокета к определенному адресу и порту компьютера функцией *bind*. Функцию *listen* и *accept* вызывать для серверных UDP-сокетов не требуется.

3. Ход работы

3.1. Получение варианта

От преподавателя был получен вариант 18.

Разработать на языке C программу transport, демонстрирующую использование основных функций работы с сокетами UNIX. Программа должна порождать четыре процесса — два клиента общаются с сервером, который является клиентом для другого сервера, которые выполняют однонаправленную передачу данных между собой через UDP сокеты, относящиеся к локальному сетевому интерфейсу lo.

3.2. Описание алгоритма

Программа была написана в соответствии с заданием. Потоки в последующем будут называться: клиент1, клиент2, клиент-сервер, сервер. Время засекается в самом начале работы программы. После того как один клиент-сервер и сервер получает определенное кол-во дейтаграмм, которое определено как

```
#define ATTEMPTS 50000
```

После этого время засекается, пакеты продолжают бесконечно и выводится помимо информации о доставке то самое, засеченное ранее, время.

3.3. Результаты измерений

```
800566
Datagram received from address: 127.0.0.1 CS Time : 0.550000 string len is:
800579
Datagram received from address: 127.0.0.1 CS Time : 0.550000 string len is:
800592
Datagram received from address: 127.0.0.1 CS Time : 0.550000 string len is:
800605
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739336
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739349
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739362
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739375
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739388
Datagram received from address: 127.0.0.1 SERVER Time : 0.160000 string len
is: 739401
```

Данные измерения показывают, что клиент-сервер обрабатывал исходное кол-во дейтаграмм в течении 55 секунд, а сервер обработал их за 16.

С первого взгляда это кажется странным, но на самом деле это объяснимо.

Ранее, в разделе Теоретических сведений было сказано, что функция recvfrom – блокирующий вызов. Допустим, что если у сервера несколько клиентов, например 100, то чтобы принимать дейтаграммы от них нужен цикл на 100 с вызовом блокирующей функции recvfrom, это приводит к тому, что мы очень много времени и ресурсов тратим на вызов данной функции. Для решения этой проблемы в ОС существуют системы неблокирующих вызовов, которые позволяют сначала оценить содержится информация о том, принял ли сокет хоть что-нибудь и затем уже выполняет чтение.

Помимо этого клиент-сервер так же отправляет сообщения серверу, который только принимает сообщения.

При запуске нескольких копий программ выскакивает ошибка в функции `bind` – это связано с тем, что порты, к которым хочет привязаться новая копия программы transport уже заняты. Но если можно было-бы увеличить кол-во клиентов, то это привело бы к ранее описанной ситуации, когда программа бегает по циклю и бесполезно растрачивает свои ресурсы. Такой сервер не являлся бы хорошим.

4. Приложение

4.1. Загрузчик – bootsect.asm

```
#ifdef _WIN32
       #define WIN32_LEAN_AND_MEAN
       #include <windows.h>
       #include <winsock2.h>
       #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 <string.h>
#include <time.h>

#define PORT1 4322
#define PORT2 4323

#define ATTEMPTS 50000

int old_time;

int init()
{
       #ifdef _WIN32
       //   Ð²Ð»Ñ   Windows   Ñ²Ð»ÐµÐ´Ñ²ÐµÐ½   Ð²Ñ²Ð·Ð²Ð°Ñ²Ñ   WSAStartup   Ð¿ÐµÑ²ÐµÐ´
Ð½Ð°Ñ²Ð°Ð»Ð¼ Ð¸Ñ²Ð¿Ð»Ñ²Ð·Ð¾Ð²Ð°Ð½Ð¸Ñ Ñ²Ð¾ÐµÐ»Ñ²Ð¾Ð²
       WSADATA wsa_data;
        return (0 == WSAStartup(MAKEWORD(2, 2), &wsa_data));
       #else
        return 1; // Ð²Ð»Ñ Ð´Ñ²Ñ²Ð³Ð¸Ñ Ð²Ð¡ Ð´ÐµÐ¹Ñ²Ñ²Ð²Ð¸Ð¹ Ð½Ð¼ Ñ²Ñ²ÐµÐ±Ñ²ÐµÐ¼Ñ²Ñ²Ñ
       #endif
}

void deinit()
{
       #ifdef _WIN32
       //  Ð²Ð»Ñ  Windows  Ñ²Ð»ÐµÐ´Ñ²ÐµÐ½  Ð²Ñ²Ð·Ð²Ð°Ñ²Ñ  WSACleanup  Ð²  ÐºÐ¾Ð½Ñ²ÐµÑ
Ñ²Ð°Ð±Ð¾Ñ²Ñ
       WSACleanup();
       #else
       // Ð²Ð»Ñ Ð´Ñ²Ñ²Ð³Ð¸Ñ Ð²Ð¡ Ð´ÐµÐ¹Ñ²Ñ²Ð²Ð¸Ð¹ Ð½Ð¼ Ñ²Ñ²ÐµÐ±Ñ²ÐµÐ¼Ñ²Ñ²Ñ
       #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, s);
    return -1;
}

void s_close(int s)
{
       #ifdef _WIN32
        closesocket(s);
```

```
        #else
         close(s);
        #endif
}

void send_request(int s, struct sockaddr_in* addr)
{
        // ÐÐ°Ð½Ð½ÑÐµ Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ DNS-Ð·Ð°Ð¿ÑÐ¾ÑÐ°. ÐÐµÑÐ°Ð»ÑÐ½Ð¾Ðµ
Ð¸Ð·ÑÑÐµÐ½Ð¸Ðµ Ð´Ð»Ñ Ð»/Ñ Ð½Ðµ ÑÑÐµÐ±ÑÐµÑÑÑ.
        char dns_datagram[] = "Client_Server";
#ifdef _WIN32
        int flags = 0;
#else
        int flags = MSG_NOSIGNAL;
#endif
        int res = sendto(s, dns_datagram, sizeof(dns_datagram), flags, (struct
sockaddr*) addr,
                        sizeof(struct sockaddr_in));
        if (res <= 0)
                sock_err("sendto", s);
}

int client()
{
        int s;
    struct sockaddr_in addr;
    int i;
    // ÐÐ½Ð¸ÑÐ¸Ð°Ð»Ð¸Ð·Ð°ÑÐ¸Ñ ÑÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð±Ð»Ð¸Ð¾ÑÐµÐºÐ¸
    init();
    // Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0)
        return sock_err("socket", s);

    // ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ ÑÑÑÑÐºÑÑÑÑ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ ÑÐ´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾
ÑÐ·Ð»Ð°
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT1); // ÐÐ¾ÑÑ DNS - 53
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    while (1) send_request(s, &addr);

    // ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑÐ°
    s_close(s);
    deinit();
    return 0;
}

int client_server()
{
        int i;
    //Socket
    int s;
    //Socket information
    struct sockaddr_in addr;

    //Input variable for sendto, recvfrom functions
    #ifdef _WIN32
        int flags = 0;
    #else
        int flags = MSG_NOSIGNAL;
    #endif

    // ÐÐ½Ð¸ÑÐ¸Ð°Ð»Ð¸Ð·Ð°ÑÐ¸Ñ ÑÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð±Ð»Ð¸Ð¾ÑÐµÐºÐ¸
    init();
    // Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0) return sock_err("socket", s);
```

```
    // ĐĐ°Đ¿Đ¾Đ»Đ½ĐµĐ½Đ¸Đµ ÑÑÑÑĐºÑÑÑÑÑ Ñ Đ°Đ´ÑĐµÑĐ¾Đ¼ ÑĐ´Đ°Đ»ĐµĐ½Đ½Đ¾Đ³Đ¾
ÑĐ·Đ»Đ°
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT1);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    //inet_aton("127.0.0.1", &addr.sin_addr.s_addr);
    //   Đ¡Đ²ÑĐ·Ñ   Đ°Đ´ÑĐµÑĐ°   Đ¸   ÑĐ¾ĐºĐµÑĐ°,   ÑÑĐ¾Đ±Ñ   Đ¾Đ½   Đ¼Đ¾Đ³
Đ¿ÑĐ¸Đ½Đ¸Đ¼Đ°ÑÑ Đ²ÑĐ¾Đ´ÑÑĐ¸Đµ Đ´ĐµĐ¹ÑĐ°Đ³ÑĐ°Đ¼Đ¼Ñ
    if (bind(s, (struct sockaddr*) &addr, sizeof(addr)) < 0)
        return sock_err("bind", s);


    char buffer[1024] = { 0 };
    int len = 0;
    int addrlen = sizeof(addr);
    // ĐÑĐ¸Đ½ÑÑĐ¸Đµ Đ¾ÑĐµÑĐµĐ´Đ½Đ¾Đ¹ Đ´ĐµĐ¹ÑĐ°Đ³ÑĐ°Đ¼Đ¼Ñ

    //sprintf(buffer, "Length of your string: %d chars.", len);
    // ĐÑĐ¿ÑĐ°Đ²ĐºĐ° ÑĐµĐ·ÑĐ»ÑÑĐ°ÑĐ°-Đ´ĐµĐ¹ÑĐ°Đ³ÑĐ°Đ¼Đ¼Ñ ĐºĐ»Đ¸ĐµĐ½ÑÑ
    //sendto(s, buffer, strlen(buffer), flags, (struct sockaddr*) &addr, addrlen);

    // ĐĐ°ĐºÑÑÑĐ¸Đµ ÑĐ¾ĐºĐµÑĐ°

    //s_close(s_server);

    int s_client;
    struct sockaddr_in addr_client;
    s_client = socket(AF_INET, SOCK_DGRAM, 0);
    if (s_client < 0)
        return sock_err("socket", s_client);
    // ĐĐ°Đ¿Đ¾Đ»Đ½ĐµĐ½Đ¸Đµ ÑÑÑÑĐºÑÑÑÑÑ Ñ Đ°Đ´ÑĐµÑĐ¾Đ¼ ÑĐ´Đ°Đ»ĐµĐ½Đ½Đ¾Đ³Đ¾
ÑĐ·Đ»Đ°
    memset(&addr_client, 0, sizeof(addr_client));
    addr_client.sin_family = AF_INET;
    addr_client.sin_port = htons(PORT2); // ĐĐ¾ÑÑ DNS - 53
    addr_client.sin_addr.s_addr = inet_addr("127.0.0.1");

    int j=0;
    //int old_time = clock();
    float t;
    while(1)
    {

        int rcv = recvfrom(s, buffer, sizeof(buffer), 0, (struct sockaddr*) &addr,
                        &addrlen);
        if (rcv > 0)
        {
            unsigned int ip = ntohl(addr.sin_addr.s_addr);
            printf("Datagram received from address: %u.%u.%u.%u ",
                (ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, (ip)& 0xFF);
            if (j == ATTEMPTS)
            {
                   int new_time = clock();
                   t = (float)(new_time - old_time)/CLOCKS_PER_SEC;
            }
            if (j > ATTEMPTS)
            {
                   printf("SERVER Time : %f", t);

            }
            for (i = 0; i < rcv; i++)
            {
                if (buffer[i] == '\0')
                    break;
                len++;
            }
            j++;
```

```
            printf(" string len is: %d\n", len);
        }


        send_request(s_client, &addr_client);
    }
    // ÐÐÐºÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐµÐ´Ð°*/

    s_close(s);
    s_close(s_client);
    deinit();
       return 0;
}

int server()
{
       int i;
       //Socket
       int s;
       //Socket information
    struct sockaddr_in addr;

    //Input variable for sendto, recvfrom functions
    #ifdef _WIN32
       int flags = 0;
       #else
           int flags = MSG_NOSIGNAL;
       #endif

       // ÐÐ½Ð¸Ñ Ð°Ð»Ð¸Ð°Ñ Ñ ÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð»Ð¸Ð¾ÑÐµÐºÐ¸
    init();
    // Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0) return sock_err("socket", s);

    // ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ Ñ Ñ Ñ Ñ ÐºÑ Ñ Ñ Ñ Ñ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ Ñ Ð´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾ Ñ Ð·Ð°Ð»Ð¸Ð¾ÑÐ·Ð¾Ð»Ð°
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT2);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    //inet_aton("127.0.0.1", &addr.sin_addr.s_addr);
    //    Ð¡Ð²Ñ Ð·Ñ    Ð°Ð´Ñ Ðµ Ñ Ð¾ÐµÐ´Ð¸Ð½ÐµÐ½Ð¸Ñ    Ð¾Ð½    Ñ Ð¾ÐºÐµÑÐ°    Ð¾Ð½Ð¾Ð³ Ð¿Ñ Ð¸Ð½Ð¸Ð°Ñ Ð²Ñ Ð¾Ð¿Ñ Ð¾Ñ Ð´ÐµÐ¹Ñ Ð°Ð³Ñ Ð°Ð¼Ð¼Ñ
    if (bind(s, (struct sockaddr*) &addr, sizeof(addr)) < 0)
        return sock_err("bind", s);


    char buffer[1024] = { 0 };
    int len = 0;
    int addrlen = sizeof(addr);
    // Ð Ñ Ð¸Ð½Ñ Ð¾Ñ Ðµ Ð¾Ñ Ð¾Ñ Ð´Ð¾Ð¾Ð¹ Ð´ÐµÐ¹Ñ Ð°Ð³Ñ Ð°Ð¼Ð¼Ñ
    int j=0;
    float t;
    while(1)
    {
        int rcv = recvfrom(s, buffer, sizeof(buffer), 0, (struct sockaddr*) &addr,
                                            &addrlen);
        if (rcv > 0)
        {
            unsigned int ip = ntohl(addr.sin_addr.s_addr);
            printf("Datagram received from address: %u.%u.%u.%u ",
                (ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, (ip)& 0xFF);

            if (j == ATTEMPTS)
            {
                   int new_time = clock();
                   t = (float)(new_time - old_time)/CLOCKS_PER_SEC;
            }
```

```
                if (j > ATTEMPTS)
                {
                       printf("CS Time : %f", t);

                }

                for (i = 0; i < rcv; i++)
                {
                    if (buffer[i] == '\0')
                        break;
                    len++;
                }
                j++;
                printf(" string len is: %d\n", len);
             }
        }
        //sprintf(buffer, "Length of your string: %d chars.", len);
        // ÐÑÐ¿ÑÐ²ÐºÐ° ÑÐµÐ·ÑÐ»ÑÑÐ°ÑÐ°-Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ ÐºÐ»Ð¸ÐµÐ½ÑÑ
        //sendto(s, buffer, strlen(buffer), flags, (struct sockaddr*) &addr, addrlen);

        // ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑÐ°
        s_close(s);
        deinit();
           return 0;
    }

    int main()
    {
           int pcl;
           old_time = clock();
           if (pcl = fork())
           {
                  client();
                  wait(pcl);
                  return 0;
           }
           if (pcl = fork())
           {
                  client();
                  wait(pcl);
                  return 0;
           }
           if (pcl = fork())
           {
                  client_server();
                  wait(pcl);
                  return 0;
           }
           return server();
}
```