1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 2**

1. по дисциплине «Операционные системы»
2. Выполнил
3. студент гр. 23508/4 Е.Г. Проценко
4. Проверил
5. преподаватель Е.Ю. Резединова
6. Санкт-Петербург
7. 2016
8. **Формулировка задания**

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

1. **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.

1. **Ход работы**
   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 cокеты, относящиеся к локальному сетевому интерфейсу lo.

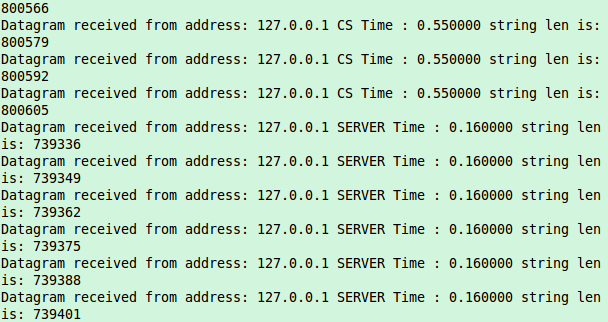
* 1. **Описание алгоритма**

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.

* 1. **Результаты измерений**



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.

1. **Приложение**
   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**

**// ÐÐ»Ñ Windows ÑÐ»ÐµÐ´ÑÐµÑ Ð²ÑÐ·Ð²Ð°ÑÑ WSAStartup Ð¿ÐµÑÐµÐ´ Ð½Ð°ÑÐ°Ð»Ð¾Ð¼ Ð¸ÑÐ¿Ð¾Ð»ÑÐ·Ð¾Ð²Ð°Ð½Ð¸Ñ ÑÐ¾ÐºÐµÑÐ¾Ð²**

**WSADATA wsa\_data;**

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

**#else**

**return 1; // ÐÐ»Ñ Ð´ÑÑÐ³Ð¸Ñ ÐÐ¡ Ð´ÐµÐ¹ÑÑÐ²Ð¸Ð¹ Ð½Ðµ ÑÑÐµÐ±ÑÐµÑÑÑ**

**#endif**

**}**

**void deinit()**

**{**

**#ifdef \_WIN32**

**// ÐÐ»Ñ Windows ÑÐ»ÐµÐ´ÑÐµÑ Ð²ÑÐ·Ð²Ð°ÑÑ WSACleanup Ð² ÐºÐ¾Ð½ÑÐµ ÑÐ°Ð±Ð¾ÑÑ**

**WSACleanup();**

**#else**

**// ÐÐ»Ñ Ð´ÑÑÐ³Ð¸Ñ ÐÐ¡ Ð´ÐµÐ¹ÑÑÐ²Ð¸Ð¹ Ð½Ðµ ÑÑÐµÐ±ÑÐµÑÑÑ**

**#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)**

**{**

**// ÐÐ°Ð½Ð½ÑÐµ Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ DNS-Ð·Ð°Ð¿ÑÐ¾ÑÐ°. ÐÐµÑÐ°Ð»ÑÐ½Ð¾Ðµ Ð¸Ð·ÑÑÐµÐ½Ð¸Ðµ Ð´Ð»Ñ Ð»/Ñ Ð½Ðµ ÑÑÐµÐ±ÑÐµÑÑÑ.**

**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;**

**// ÐÐ½Ð¸ÑÐ¸Ð°Ð»Ð¸Ð°Ð·Ð°ÑÐ¸Ñ ÑÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð±Ð»Ð¸Ð¾ÑÐµÐºÐ¸**

**init();**

**// Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°**

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

**if (s < 0)**

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

**// ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ ÑÑÑÑÐºÑÑÑÑ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ ÑÐ´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾ ÑÐ·Ð»Ð°**

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

**addr.sin\_family = AF\_INET;**

**addr.sin\_port = htons(PORT1); // ÐÐ¾ÑÑ DNS - 53**

**addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");**

**while (1) send\_request(s, &addr);**

**// ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑÐ°**

**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**

**// ÐÐ½Ð¸ÑÐ¸Ð°Ð»Ð¸Ð°Ð·Ð°ÑÐ¸Ñ ÑÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð±Ð»Ð¸Ð¾ÑÐµÐºÐ¸**

**init();**

**// Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°**

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

**if (s < 0) return sock\_err("socket", s);**

**// ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ ÑÑÑÑÐºÑÑÑÑ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ ÑÐ´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾ ÑÐ·Ð»Ð°**

**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);**

**// Ð¡Ð²ÑÐ·Ñ Ð°Ð´ÑÐµÑÐ° Ð¸ ÑÐ¾ÐºÐµÑÐ°, ÑÑÐ¾Ð±Ñ Ð¾Ð½ Ð¼Ð¾Ð³ Ð¿ÑÐ¸Ð½Ð¸Ð¼Ð°ÑÑ Ð²ÑÐ¾Ð´ÑÑÐ¸Ðµ 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ÑÐ¸Ðµ Ð¾ÑÐµÑÐµÐ´Ð½Ð¾Ð¹ Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ**

**//sprintf(buffer, "Length of your string: %d chars.", len);**

**// ÐÑÐ¿ÑÐ°Ð²ÐºÐ° ÑÐµÐ·ÑÐ»ÑÑÐ°ÑÐ°-Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ ÐºÐ»Ð¸ÐµÐ½ÑÑ**

**//sendto(s, buffer, strlen(buffer), flags, (struct sockaddr\*) &addr, addrlen);**

**// ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑÐ°**

**//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);**

**// ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ ÑÑÑÑÐºÑÑÑÑ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ ÑÐ´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾ ÑÐ·Ð»Ð°**

**memset(&addr\_client, 0, sizeof(addr\_client));**

**addr\_client.sin\_family = AF\_INET;**

**addr\_client.sin\_port = htons(PORT2); // ÐÐ¾ÑÑ 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);**

**}**

**// ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑ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**

**// ÐÐ½Ð¸ÑÐ¸Ð°Ð»Ð¸Ð°Ð·Ð°ÑÐ¸Ñ ÑÐµÑÐµÐ²Ð¾Ð¹ Ð±Ð¸Ð±Ð»Ð¸Ð¾ÑÐµÐºÐ¸**

**init();**

**// Ð¡Ð¾Ð·Ð´Ð°Ð½Ð¸Ðµ UDP-ÑÐ¾ÐºÐµÑÐ°**

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

**if (s < 0) return sock\_err("socket", s);**

**// ÐÐ°Ð¿Ð¾Ð»Ð½ÐµÐ½Ð¸Ðµ ÑÑÑÑÐºÑÑÑÑ Ñ Ð°Ð´ÑÐµÑÐ¾Ð¼ ÑÐ´Ð°Ð»ÐµÐ½Ð½Ð¾Ð³Ð¾ ÑÐ·Ð»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ÐµÑÐ° Ð¸ ÑÐ¾ÐºÐµÑÐ°, ÑÑÐ¾Ð±Ñ Ð¾Ð½ Ð¼Ð¾Ð³ Ð¿ÑÐ¸Ð½Ð¸Ð¼Ð°ÑÑ Ð²ÑÐ¾Ð´ÑÑÐ¸Ðµ 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ÑÐ¸Ðµ Ð¾ÑÐµÑÐµÐ´Ð½Ð¾Ð¹ 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);**

**// ÐÑÐ¿ÑÐ°Ð²ÐºÐ° ÑÐµÐ·ÑÐ»ÑÑÐ°ÑÐ°-Ð´ÐµÐ¹ÑÐ°Ð³ÑÐ°Ð¼Ð¼Ñ ÐºÐ»Ð¸ÐµÐ½ÑÑ**

**//sendto(s, buffer, strlen(buffer), flags, (struct sockaddr\*) &addr, addrlen);**

**// ÐÐ°ÐºÑÑÑÐ¸Ðµ ÑÐ¾ÐºÐµÑÐ°**

**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();**

**}**