Лабораторная работа по курсу "Операционные системы"

Межпроцессное взаимодействие в ОС Linux: сокеты

Цель работы: Научиться использовать сокеты UDP и TCP для взаимодействия процессов в сети.

1. Взаимодействие между клиентом и сервером при помощи сокетов UDP

В файлах **udpserver.c** и **udpclient.c** находятся программы эхо-сервера и эхо-клиента для сокетов UDP. Клиент посылает серверу запрос – строку текста, введенную из *stdin*, сервер выводит ее на *stdout* вместе с информацией об адресе клиента, и отсылает строку обратно клиенту. Клиент принимает ответ от сервера и распечатывает его. Затем клиент ожидает ввода следующей строки. При считывании *Ctrl-d* клиент завершается. IP-адрес сервера задается в параметре командной строки при запуске клиента. Номер порта сервера может быть любым, но он должен быть заранее известен клиенту. Он задается в программах сервера и клиента константой **SERV_PORT**.

Задание 1.

1.1. Выполните программы **udpserver.c** и **udpclient.c**, запустив их из разных окон компьютера, сначала запустите **udpserver.c**. При запуске клиента задайте в командной строке один параметр: адрес сервера **127.0.0.1** (это адрес петли обратной связи для локальной отладки сетевых приложений). *Что произойдет, если сервер запустить позже клиента; если завершить сервер во время работы клиента (поясните в отчете)?*

1.2. В программе **udpserver.c** перед отправкой ответа клиенту добавьте небольшую задержку (10 с), имитирующую процесс обработки запроса. Запустите одновременно несколько эхо-клиентов из разных окон. *Нарисуйте в отчете временные диаграммы сервера при обслуживании нескольких клиентов.*

2. Взаимодействие между клиентом и сервером при помощи сокетов TCP

В файлах **tcpserver.c** и **tcpclient.c** находятся программы сервера и клиента для сокетов TCP. Клиент посылает серверу строкуи текста, введенные из *stdin*, сервер получает их и выводит на *stdout*. IP-адрес и номер порта сервера задаются в двух параметрах командной строки при запуске клиента. Номер порта сервера может быть любым, но он должен быть заранее известен клиенту. Он задается в программе сервера константой **SERV_PORT**.

Задание 2.

2.1. Выполните начальную версию программ **tcpserver.c** и **tcpclient.c**, запустив их из разных окон компьютера, сначала запустите **tcpserver.c**. В качестве номера порта сервера задайте **30 000 плюс номер Вашего ПК** (1 – 28). Этот номер следует задать в константе **SERV_PORT** в программе сервера перед его компиляцией. При запуске клиента задайте в командной строке два параметра: адрес сервера **127.0.0.1** и номер порта сервера (30 000 плюс номер Вашего ПК). *Что произойдет, если сервер запустить позже клиента; если завершить сервер во время работы клиента (поясните в отчете)?*

Во время работы программ получите информацию об активных сокетах при помощи команды

$netstat -a -p

Первая группа сокетов, выводимая по этой команде, – это сокеты Интернет, вторая группа – сокеты UNIX. Найдите свои сокеты в первой группе. Колонка **Local Address** содержит адрес локального конца сокета (на Вашем компьютере), колонка **Foreign Address** содержит адрес удаленного конца сокета (на компьютере, с которым взаимодействует Ваш сокет). Адрес состоит из двух частей: **IP-адрес.порт**. *Звездочка* в первой части адреса **Local Address** Вашего прослушивающего сокета обозначает **INADDR_ANY**, т.е. универсальный IP-адрес, и означает, что сервер будет принимать соединения, поступающие для любого IP-адреса данного компьютера и для номера порта сервера, указанного после точки (Это номер, заданный в **SERV_PORT**.). Звездочки в обеих частях адреса **Foreign Address** Вашего прослушивающего сокета обозначают

универсальный IP-адрес и любой порт, и означают, что сервер будет принимать соединения от клиента с любым IP-адресом и любым номером порта. *Выпишите в отчет информацию о Ваших сокетах*. *Объясните в отчете*, когда и какими функциями они были созданы, в каких состояниях находятся, какова пара адресов для каждого из сокетов? При выполнении клиента и сервера на разных компьютерах, сколько сокетов Вы увидите на каждом компьютере?

2.2. Выполните программы клиента и сервера на разных компьютерах. Для этого следует запустить сервер на своем компьютере, а программу клиента – с другого компьютера, указав в командной строке IP-адрес Вашего компьютера (ниже объясняется, как определить IP-адрес) и номер порта сервера на Вашем компьютере (30 000 + номер Вашего ПК).
Определение IP-адреса компьютера. Получите информацию о сетевых интерфейсах Вашего компьютера при помощи команды **$netstat -nie** или **$ifconfig -a**
Вы должны увидеть два интерфейса: **lo** (интерфейс обратной связи, или закольцовки) и **eth0** (локальная сеть *Ethernet*) со следующей информацией об интерфейсах: **inet addr** – IP-адрес Вашего компьютера, **Linc encap** – используемая технология канального уровня, **Bcast** – широковещательный IP-адрес, **Hwaddr** – физический адрес сетевой карты, **Mask –** маска подсети, **MTU** – максимальный размер кадра.

2.3. Модифицируйте программу сервера (дайте ей имя **tcpserver2.c**): 1) перед выводом запроса клиента на экран добавьте небольшую задержку (10 с), имитирующую процесс обработки запроса; 2) распечатывайте информацию об адресе клиента (его IP-адрес и номер порта). Запустите сервер в одном окне, а из других окон одновременно запустите несколько клиентов.

2.4. Модифицируйте сервер (имя **tcpserver3.c**): сделайте Ваш последовательный сервер параллельным, обслуживая каждого клиента в отдельном процессе сервера. Для этого после **accept()** создайте дочерний процесс, в который переместите те строки кода, в которых сервер выводит сообщение об установлении соединения, читает из сокета полученные данные, ожидает 10 с, выводит данные на экран и закрывает присоединенный сокет. Кроме этого, в начале дочернего процесса закройте прослушиваемый сокет, а в конце дочернего процесса добавьте **exit(0)**. Основной цикл родительского сервера должен включать: вычисление **clilen**, **accept()**, **fork()** и закрытие присоединенного сокета. Обязательно удаляйте завершившиеся дочерние процессы, обрабатывая сигнал SIGCHLD. Выполните параллельный сервер с несколькими одновременно запущенными клиентами. *Нарисуйте в отчете временные диаграммы работы последовательного и параллельного серверов при обслуживании нескольких клиентов.*

Порядок выполнения лабораторной работы

1. Выполните задания 1 – 2.
2. Занесите в отчет описание заданий со всеми требуемыми пояснениями.

Требования

1. При подготовке к лабораторной работе (дома) запишите в конспект:
   а) тексты программ сервера из задания 2.4 с комментариями;
   б) описание структуры **sockaddr_in**, описание функций работы с сокетами, включая использованные функции преобразования адресов.
2. Студент должен знать ответы на следующие вопросы:

Вопросы

1. Какие типы сокетов используются в ОС UNIX?
2. Каково назначение и параметры системного вызова **socket()**? **connect()**?
3. Каково назначение и параметры системного вызова **bind()**? Обязательно ли его вызывать для сервера? Для клиента? Почему?
4. Каково назначение и параметры вызова **accept()**? Какой результат возвращает **accept()**?
5. Какова последовательность действий, выполняемых сервером в **udpserver.c** и **tcpserver.c**?
6. Какова последовательность действий, выполняемых клиентом в **udpclient.c** и **tcpclient.c**?
7. Что произойдет, если в параллельном сервере TCP не закрывать в дочернем сервере прослушиваемый сокет, а в родительском сервере – присоединенный сокет? Почему?
8. Какие другие способы построения параллельного сервера Вы можете предложить?

9. Почему необходимо преобразование адреса сокета в сетевой порядок следования байтов?

Источники информации

1. Стивенс У. UNIX: разработка сетевых приложений. – СПб.: Питер, 2004. (Главы 2–6)
2. Камер Д.Э., Стивенс Д.Л. Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX. – М.: Издательский дом "Вильямс", 2002. (Главы 4–8, 10, 11)
3. Ш. Уолтон. Создание сетевых приложений в среде Linux. – М.: Издательский дом "Вильямс", 2001. (Глава 4)

ПРИЛОЖЕНИЕ. Тексты программ

```
/*          udpserver.c                */
#include <sys/socket.h>
#include <sys/types.h>
#include <stdio.h>
#include <netinet/in.h>
#include <sys/stat.h>
#include <unistd.h>

#define SERV_PORT 20001             /*    Номер порта сервера                 */
#define BUFSIZE 1024
#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main() {
int sockfd, n, len;
char mesg[BUFSIZE], ipadr[16];
struct sockaddr_in servaddr;        /*    Для адреса сервера                  */
struct sockaddr_in cliaddr;         /*    Для адреса клиента                  */

if ((sockfd=socket(AF_INET, SOCK_DGRAM, 0))<0)/* Создать сокет типа SOCK_DGRAM */
      {perror("socket problem"); exit(1);}

memset(&servaddr, 0, SLEN);         /*    Обнулить структуру servaddr         */
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY); /*  Адрес сервера - любой
            IP адрес этого компьютера, преобразовать его в сетевой порядок    */
servaddr.sin_port = htons(SERV_PORT); /*  Преобразование номера порта         */
                  /*          Привязать сокет к адресу и порту сервера        */
if (bind(sockfd, (SADDR *) &servaddr, SLEN)<0)
      {perror("bind problem"); exit(1);}
printf("SERVER starts...\n");

for (;;) {              /*          ГЛАВНЫЙ ЦИКЛ СЕРВЕРА                      */
      len = SLEN;
            /*    Чтение очередного запроса от очередного клиента           */
      if ((n = recvfrom(sockfd, mesg, BUFSIZE, 0, (SADDR *)&cliaddr, &len)) < 0)
            {perror("recvfrom"); exit(1);}
      mesg[n] = 0;
            /*    Печать текста запроса и адреса клиента                    */
      printf("REQUEST %s      FROM %s : %d\n", mesg, inet_ntop(AF_INET,
            (void *)&cliaddr.sin_addr.s_addr, ipadr,16), ntohs(cliaddr.sin_port));
            /*    Посылка ответа клиенту                                    */
      if (sendto(sockfd, mesg, n, 0, (SADDR *)&cliaddr, len) < 0)
            {perror("sendto"); exit(1);}
}
}

/*       udpclient.c          Запуск:  udpclient IP_адрес_сервера            */
#include <sys/socket.h>
#include <sys/types.h>
#include <stdio.h>
#include <netinet/in.h>
#include <sys/stat.h>
```

```
#include <unistd.h>

#define SERV_PORT 20001                    /*    Номер порта сервера          */
#define BUFSIZE 1024
#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main(int argc, char **argv) {
int sockfd, n;
char sendline[BUFSIZE], recvline[BUFSIZE + 1];
struct sockaddr_in servaddr;               /*    Для адреса сервера           */
struct sockaddr_in cliaddr;                /*    Для адреса клиента           */

if (argc != 2) {printf("usage: client <IPaddress of server>\n"); exit(1);}
                  /*    Сформировать структуру с адресом сервера             */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(SERV_PORT);
      /* Преобразовать IP-адрес сервера в сетевой порядок и записать в servaddr */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr)<0)
      {perror("inet_pton problem"); exit(1);}

if ((sockfd=socket(AF_INET, SOCK_DGRAM, 0))<0) /* Создать сокет типа SOCK_DGRAM */
      {perror("socket problem"); exit(1);}

write(1,"Enter string\n", 13);             /*    Печать приглашения           */
while ((n=read(0, sendline, BUFSIZE)) > 0) { /*       Читать строку с stdin   */
                                           /*    Отослать строку серверу      */
      if (sendto(sockfd, sendline, n, 0, (SADDR *)&servaddr, SLEN) == -1)
            {perror("sendto problem"); exit(1);}
                                           /*    Принять сообщение от сервера */
      if (recvfrom(sockfd, recvline, BUFSIZE, 0, NULL, NULL) == -1)
            {perror("recvfrom problem"); exit(1);}
                                           /*    и распечатать его            */
      printf("REPLY FROM SERVER= %s\n", recvline);
}
close(sockfd);                             /*          Закрыть сокет          */
}

/*           tcpserver.c            */
#include <sys/socket.h>
#include <sys/types.h>
#include <stdio.h>
#include <netinet/in.h>

#define SERV_PORT ...                      /*    Номер порта сервера          */
#define BUFSIZE 100
#define SADDR struct sockaddr
#define SIZE sizeof(struct sockaddr_in)

int main() {
int lfd, cfd;
int nread, clilen;
char buf[BUFSIZE];
struct sockaddr_in servaddr;               /*    Для адреса сервера           */
struct sockaddr_in cliaddr;                /*    Для адреса клиента           */

if ((lfd = socket(AF_INET, SOCK_STREAM, 0))<0)/*Создать сокет типа SOCK_STREAM*/
      {perror("socket"); exit(1);}

memset(&servaddr, 0, SIZE);   /*  Обнулить структуру для адреса сервера      */
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);   /*  Адрес сервера - любой
            IP адрес этого компьютера, преобразовать его в сетевой порядок    */
servaddr.sin_port = htons(SERV_PORT); /* Порт сервера,преобр.в сетев.порядок  */
```

```
                  /*          Привязать сокет к адресу и порту сервера        */
if (bind(lfd, (SADDR *) &servaddr, SIZE)<0)
      {perror("bind"); exit(1);}
                  /*                Сделать сокет lfd прослушиваемым          */
if (listen(lfd, 5) < 0) {perror("listen"); exit(1);}

for(;;){    /*                ГЛАВНЫЙ ЦИКЛ СЕРВЕРА                            */
      clilen =SIZE;
      /* Ожидаем соединения, в cfd получим дескриптор присоединенного сокета,
         в cliaddr - адрес клиента, в clilen - действит. число байтов адреса  */
      if ((cfd = accept(lfd, (SADDR *)&cliaddr, &clilen)) < 0)
            {perror("accept"); exit(1);}
            /* Соединение установлено, информация о клиенте - в cliaddr       */
      printf("connection established\n");

            /* Прочитать сообщение от клиента и вывести его в stdout          */
      while ((nread = read(cfd, buf, BUFSIZE))> 0)
            write(1, &buf, nread);
      if (nread == -1){perror("read"); exit(1);}
      close(cfd);                   /*    закрыть присоединенный сокет        */
}           /*          КОНЕЦ ГЛАВНОГО ЦИКЛА СЕРВЕРА                          */
}

/*   tcpclient.c        Запуск:  tcpclient IP_адрес_сервера порт_сервера      */
#include <sys/socket.h>
#include <sys/types.h>
#include <stdio.h>
#include <netinet/in.h>
#define BUFSIZE 100
#define SADDR struct sockaddr
#define SIZE sizeof(struct sockaddr_in)

int main(int argc, char *argv[]){
int fd;
int nread;
char buf[BUFSIZE];
struct sockaddr_in servaddr;
if (argc < 3) {printf("Too few arguments \n"); exit(1);}
if ((fd = socket(AF_INET, SOCK_STREAM, 0))<0) /*Создать сокет типа SOCK_STREAM*/
      {perror("socket creating"); exit(1);}
      /*          Сформировать структуру с адресом сервера                    */
memset(&servaddr, 0, SIZE);         /*  Обнулить структуру для адреса         */
servaddr.sin_family = AF_INET;
      /* Преобразовать адрес сервера в двоичное число в сетевом порядке       */
if(inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
      {perror("bad address"); exit(1);}
      /*    Преобразовать номер порта сервера в сетевой порядок               */
servaddr.sin_port = htons(atoi(argv[2]));

      /*    Установить соединение с сервером, указанным в servaddr            */
if (connect(fd, (SADDR *)&servaddr, SIZE)<0)
      {perror("connect"); exit(1);}

write(1, "Input message to send\n", 22);  /*    Вывод приглашения на экран    */
while ((nread = read(0, buf, BUFSIZE)) > 0)     /*    Чтение строки из stdin   */
      if (write(fd, buf, nread)<0)              /*    Посылка строки серверу   */
            {perror("write"); exit(1);}
close(fd);                                    /*    Закрыть сокет            */
exit(0);
}
```
