Министерство образования и науки РФ

Томский государственный университет систем управления и радиоэлектроники (ТУСУР)
Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

А.С. Романов

Системное программирование

Тема № 7 – Программирование сокетов

Томск 2014

Цель работы

Познакомиться с основными аспектами работы с сокетами. Познакомиться с соответствующими функциями WinAPI и POSIX API.

Краткие теоретические сведения

1. Общие сведения о сокетах

Сокеты – одно из средств межпроцессного взаимодействия (IPC). Сокет представляет собой один конец двусторонней связи, между двумя программами. Соединяя два сокета, можно передавать данные между разными процессами как в рамках одной системы, так и между процессами, запущенными на разных машинах в сети. Кроме того, с их помощью можно организовать взаимодействие с программами, работающими под управлением других операционных систем.

Реализация сокетов осуществляет инкапсуляцию протоколов сетевого и транспортного уровней.

Сокет состоит из IP адреса машины и номера порта, используемого приложением. Уникальность IP адресов в сети Интернет и порта в рамках одной машины делает сам сокет уникальным в сети Интернет.

При взаимодействии двух процессов, использующих сокеты, можно выделить серверную часть и клиентскую: серверный процесс инициализирует сокет и ждёт входящих соединений от других процессов, а клиентский процесс инициализирует соединение с сервером.

Существует два типа сокетов **потоковые** и **датаграммные**.

Потоковый сокет – это сокет с установлением соединения, состоящий из потока байтов, который может быть двунаправленным, то есть через конечную точку может передавать и получать данные. Потоковый сокет осуществляет надежную передачу, подходит для передачи больших объемов данных. Для передачи используется протокол TCP.

Датаграммные сокеты – сокеты без установления соединения. Используется протокол UDP. По сравнению с потоковым сокетом обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться в пути, дублироваться и переупорядочиваться.

В случае **локального** варианта взаимодействия через сокеты обмен данными происходит через специальные файлы, расположенные в файловой системе и, фактически, является расширением идеи именованных каналов, но с интерфейсом сокетов.

Рассмотрим, какие средства для работы с сокетами существуют в операционных системах Unix/Linux

2. Потоковые сокеты

Алгоритм работы клиентского и серверного приложений, использующих потоковые сокеты, и соответствующие API функции представлены на рис. 1. Рассмотрим этапы и функции подробнее.

Все объявления функций и типов, относящиеся к сокетам, можно найти в файле socket.h.

Для создания сокета используется функция ***socket()***:

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (      // создание сокета
  int domain,     // домен и семейство адресов: AF_UNIX и AF_INET
  int type,       // тип сокета: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  int protocol    // протокол
);
```

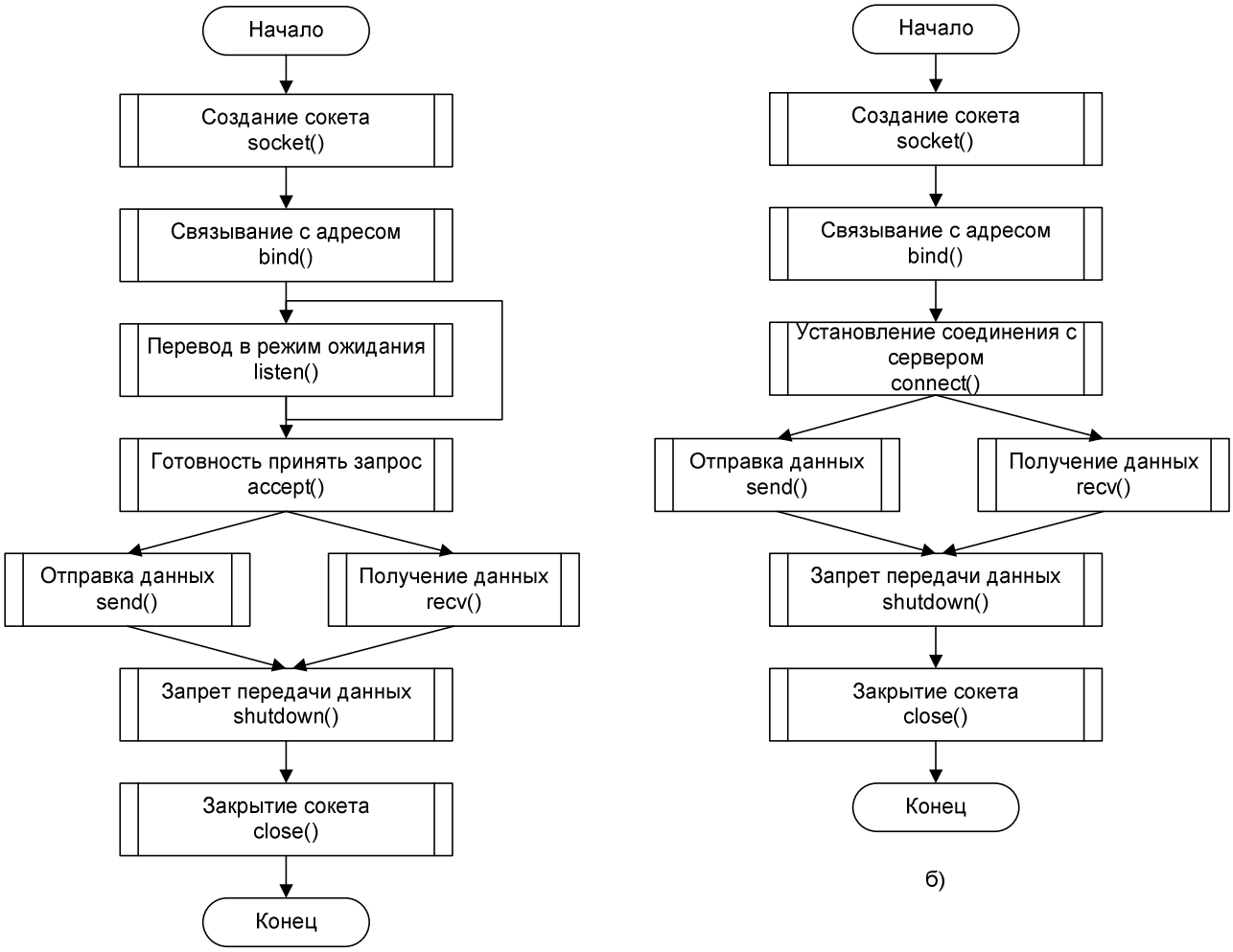

Рисунок 1 – Алгоритм установления связи и обмена данными между клиентом (б) и сервером (а) на основе потокового сокета

Сокет имеет три атрибута:

1) домен - чаще других используются домены Unix и Internet, которые задаются константами AF_UNIX, AF_INET (для протокола IPv4) и AF_INET6 (для протокола IPv6) соответственно.

2) тип – определяет способ передачи данных по сети. SOCK_STREAM – потоковый сокет, SOCK_DGRAM – датаграммный, SOCK_RAW используется для низкоуровневой работы с протоколами IP, ICMP.

3) протокол – как правило, определяется автоматически в зависимости от типа сокета и домена - в этом случае передается 0.

Для связи сокета с адресом используется функция ***bind()***:

```
#include <sys/types.h>
#include <sys/socket.h>
int bind ( // связывание сокета сервера с адресом
  int sockfd, // дескриптор сокета
  struct sockaddr *addr, // указатель на структуру с адресом
  int addrlen // длина структуры адреса
);
```

Первый параметр функции - это дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр содержит указатель на структуру с адресом, а третий - длину этой структуры.

Процедуры создания сокета и связывания его с адресом являются общими для клиента и для сервера. Перейдем к рассмотрению специфики.

Серверный сокет нужно перевести в режим ожидания соединений от клиентов. Делается это с помощью функции ***listen()***, принимающей на входе дескриптор очереди и размер очереди запросов:

```
int listen ( // перевод сокета сервера в режим ожидания запросов со стороны
клиентов
  int sockfd, // дескриптор сокета
  int backlog // размер очереди запросов
);
```

В случае если клиентский сокет пытается соединиться с серверным, а у последнего вся очередь заполнена, запрос игнорируется.

Когда сервер готов принять запрос, он вызывает функцию ***accept()***:

```
int accept ( //  сервер создаёт для общения с клиентом новый сокет, когда сервер
готов принять запрос
  int sockfd, // дескриптор слушающего сокета
  void *addr, // адрес сокета клиента
  int *addrlen // длина
);
```

Функция создает новый сокет и возвращает дескриптор этого сокета. При этом адрес нового сокета такой же, как у слушающего сокета.

Установка соединения со стороны клиента инициализируется с помощью функции ***connect()***:

```
int connect ( //  установления соединения клиентов с сервером
  int sockfd, // сокет, использующийся для обмена данными с сервером
  struct sockaddr *serv_addr, // указатель на структуру с адресом сервера
  int addrlen // длина этой структуры
);
```

Первый параметр задает сокет, который будет использоваться для обмена данных с сервером, второй – указатель на структуру данных, содержащую адрес сервера, третий – длина этой структуры. Клиентскому сокету можно предварительно присвоить порт, вызвав функцию bind(), в противном случае порт будет выбран автоматически.

После того как инициализированы клиентский и серверный сокеты, можно начинать обмен данными. Для этого предназначены функции ***send()*** (отправка данных) и ***recive()*** (получение данных):

```
int send ( // отправка данных
  int sockfd, // дескриптор сокета, через который отправляются данные
  const void *msg, // указатель на буфер с данными
  int len, // длина буфера данных
  int flags // набор битовых флагов
);
```

```
int recv( // чтение данных из сокета
int sockfd, // дескриптор сокета, из которого считываются данные
void *buf, // указатель на буфер
int len, // длина буфера данных
int flags // набор битовых флагов
);
```

Обе функции получают на входе дескриптор сокета, указатель на буфер и набор флагов. Функция send() возвращает количество отправленных данных (в байтах) или -1 в случае ошибки, recv() возвращает количество полученных данных (в байтах), 0 в случае разрыва соединения и -1 в случае ошибки.

Запретить передачу данных в одном направлении можно с помощью функции ***shutdown()***, конкретизировав что именно вы хотите запретить (чтение, запись, чтение и запись) с помощью параметра how:

```
int shutdown (     // запрет передачи данных в одном направлении
  int sockfd,      // дескриптор сокета
  int how          // 0 - запрет чтения, 1 - записи, 2 - чтения и записи.
);
```

Закрытие сокета происходит с помощью ***close()***:

```
int close (        // закрытие сокета
  int fd           // дескриптор сокета
);
```

3. Датаграммные сокеты

Процедуры создания сокета и связывания для датаграммного и потокового сокета идентичны. Создав датаграммный сокет, его можно сразу использовать для отправки и получения данных. Соединение устанавливать не требуется. Закрытие датаграммного сокета осуществляется функцией close() (см. рис. 2).

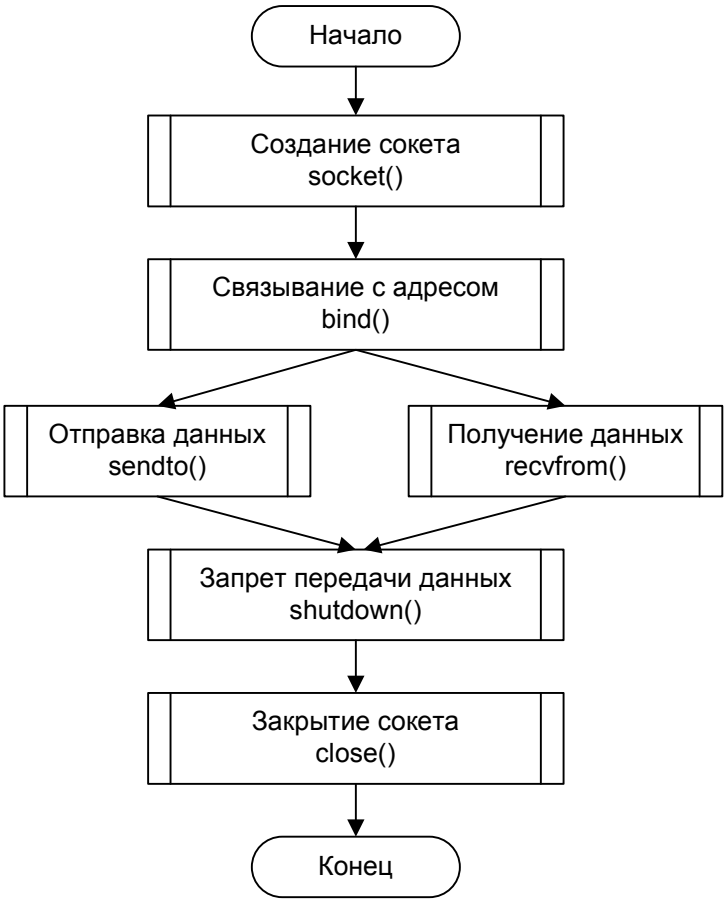

Рисунок 2 – Алгоритм работы датаграммного сокета

Для отправки данных используется функция ***sendto()***, для получения данных используется функция ***recvfrom()***. Параметры функций:

```
int sendto (            // отправка данных через датаграммный сокет
  int sockfd,           // дескриптор сокета
  const void *msg,      // указатель на буфер с данными
  int len,              // длина буфера данных
  unsigned int flags,   // набор битовых флагов
  const struct sockaddr *to,  // указатель на структуру с адресом
  int tolen                   // длина структуры адреса
);
```

```
int recvfrom (          // получение данных из датаграммного сокета
  int sockfd,           // дескриптор сокета
  void *buf,            // указатель на буфер с данными
  int len,              // длина буфера данных
  unsigned int flags,   // набор битовых флагов
  struct sockaddr *from,// указатель на структуру с адресом
  int *fromlen          // длина структуры адреса
);
```

4. Примеры использования сокетов

Пример простого клиент-серверного приложения для интернет домена на основе потоковых сокетов приведен в листинге 1 (сервер) и листинге 2 (клиент).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno;
     socklen_t clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
```

```
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd,
                 (struct sockaddr *) &cli_addr,
                 &clilen);
     if (newsockfd < 0)
          error("ERROR on accept");
     bzero(buffer,256);
     n=recv(newsockfd,buffer,255, 0);

     if (n < 0) error("ERROR reading from socket");
     printf("Here is the message: %s\n",buffer);
     n=send(newsockfd, "I got your message", 18, 0);
     if (n < 0) error("ERROR writing to socket");
     close(newsockfd);
     close(sockfd);
     return 0;
}
```

Листинг 1 - Пример простого TCP сервера в интернет домене

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
```

```
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n=send(sockfd, "I got your message", 18, 0);
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n=recv(sockfd,buffer,255, 0);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    close(sockfd);
    return 0;
}
```

Листинг 2 - Пример простого TCP клиента в интернет домене

Программы нужно откомпилировать каждую отдельно и запустить. В качестве параметра серверу передать номер порта от 2000 до 65535:

```
$ ./server 5555
```

Для запуска клиента потребуется указать имя компьютера в сети и порт. Например:

```
$ ./client computer 5555
```

После ввода сообщения на клиентской машине, оно должно отобразиться на серверной.

В случае если программы работают на одной машине, необходимо использовать UNIX домен (AF_UNIX) и соответствующую структуру для адреса:

```
Struct sockaddr_un
{
 short sun_family;      // AF_UNIX
 char sun_path[108];    // стандартная форма пути для файловой системы unix
};
```

В этом случае сокет идентичен именованному каналу (FIFO pipe) и выглядит в системе как файл нулевой длины.

Полный пример сервера и клиента представлен в листингах 3 и 4 соответственно.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/un.h>
#include <stdio.h>
void error(const char *);
int main(int argc, char *argv[])
{
   int sockfd, newsockfd, servlen, n;
   socklen_t clilen;
   struct sockaddr_un  cli_addr, serv_addr;
   char buf[80];

   if ((sockfd = socket(AF_UNIX,SOCK_STREAM,0)) < 0)
       error("creating socket");
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sun_family = AF_UNIX;
```

```
   strcpy(serv_addr.sun_path, argv[1]);
   servlen=strlen(serv_addr.sun_path) +
                    sizeof(serv_addr.sun_family);
   if(bind(sockfd,(struct sockaddr *)&serv_addr,servlen)<0)
       error("binding socket");

   listen(sockfd,5);
   clilen = sizeof(cli_addr);
   newsockfd = accept(
        sockfd,(struct sockaddr *)&cli_addr,&clilen);
   if (newsockfd < 0)
        error("accepting");
   n=read(newsockfd,buf,80);
   printf("A connection has been established\n");
   write(1,buf,n);
   write(newsockfd,"I got your message\n",19);
   close(newsockfd);
   close(sockfd);
   return 0;
}

void error(const char *msg)
{
    perror(msg);
    exit(0);
}
```

Листинг 3 - Пример простого TCP сервера в UNIX домене

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
void error(const char *);

int main(int argc, char *argv[])
{
   int sockfd, servlen,n;
   struct sockaddr_un  serv_addr;
   char buffer[82];

   bzero((char *)&serv_addr,sizeof(serv_addr));
   serv_addr.sun_family = AF_UNIX;
   strcpy(serv_addr.sun_path, argv[1]);
   servlen = strlen(serv_addr.sun_path) +
                sizeof(serv_addr.sun_family);
   if ((sockfd = socket(AF_UNIX, SOCK_STREAM,0)) < 0)
       error("Creating socket");
   if (connect(sockfd, (struct sockaddr *)
                        &serv_addr, servlen) < 0)
       error("Connecting");
   printf("Please enter your message: ");
   bzero(buffer,82);
   fgets(buffer,80,stdin);
   write(sockfd,buffer,strlen(buffer));
   n=read(sockfd,buffer,80);
   printf("The return message was\n");
   write(1,buffer,n);
   close(sockfd);
   return 0;
}
```

```
void error(const char *msg)
{
    perror(msg);
    exit(0);
}
```

Листинг 4 - Пример простого TCP клиента в Unix домене

Обратите внимание, что в двух последних примерах используются системные вызовы ***read()*** и ***write()*** для чтения и записи в сокет. Этот способ также допустим, но имеет ряд ограничений, поэтому злоупотреблять им не стоит.

Пример датаграммного сервера и простого UDP клиента для интернет домена представлен в листингах 5 и 6.

```
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
   int sock, length, n;
   socklen_t fromlen;
   struct sockaddr_in server;
   struct sockaddr_in from;
   char buf[1024];

   if (argc < 2) {
      fprintf(stderr, "ERROR, no port provided\n");
      exit(0);
   }

   sock=socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) error("Opening socket");
   length = sizeof(server);
   bzero(&server,length);
   server.sin_family=AF_INET;
   server.sin_addr.s_addr=INADDR_ANY;
   server.sin_port=htons(atoi(argv[1]));
   if (bind(sock,(struct sockaddr *)&server,length)<0)
       error("binding");
   fromlen = sizeof(struct sockaddr_in);
   while (1) {
       n = recvfrom(sock,buf,1024,0,(struct sockaddr *)&from,&fromlen);
       if (n < 0) error("recvfrom");
       write(1,"Received a datagram: ",21);
       write(1,buf,n);
       n = sendto(sock,"Got your message\n",17,
                  0,(struct sockaddr *)&from,fromlen);
       if (n  < 0) error("sendto");
   }
```

```
   return 0;
 }
```

Листинг 5 – Пример датаграммного сервера

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void error(const char *);
int main(int argc, char *argv[])
{
   int sock, n;
   unsigned int length;
   struct sockaddr_in server, from;
   struct hostent *hp;
   char buffer[256];

   if (argc != 3) { printf("Usage: server port\n");
                    exit(1);
   }
   sock= socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) error("socket");

   server.sin_family = AF_INET;
   hp = gethostbyname(argv[1]);
   if (hp==0) error("Unknown host");

   bcopy((char *)hp->h_addr,
        (char *)&server.sin_addr,
         hp->h_length);
   server.sin_port = htons(atoi(argv[2]));
   length=sizeof(struct sockaddr_in);
   printf("Please enter the message: ");
   bzero(buffer,256);
   fgets(buffer,255,stdin);
   n=sendto(sock,buffer,
            strlen(buffer),0,(const struct sockaddr *)&server,length);
   if (n < 0) error("Sendto");
   n = recvfrom(sock,buffer,256,0,(struct sockaddr *)&from, &length);
   if (n < 0) error("recvfrom");
   write(1,"Got an ack: ",12);
   write(1,buffer,n);
   close(sock);
   return 0;
}

void error(const char *msg)
{
    perror(msg);
    exit(0);
}
```

Листинг 6 – Пример простого UDP-клиента

Параметры запуска примеров программ 5 и 6 аналогичны тем, что использовались при запуске 1 и 2 программ.

Задание

1. Изучить краткие теоретические сведения и лекционный материал по теме практического задания.
2. Реализовать приведенные примеры программ.
3. Самостоятельно изучить средства программирования сокетов в ОС Windows и отразить в отчете.
4. Реализовать примеры клиентских программ под ОС Windows для обмена сообщениями с серверами TCP и UDP для Unix/Linux.
5. Написать отчет и защитить у преподавателя.

Варианты индивидуальных заданий

Отсутствуют.

Контрольные вопросы

1. Что такое сокет?
2. Какие бывают сокеты, в чем их особенности?
3. Какие атрибуты есть у сокета?
4. Особенность приложения клиент – сервер, основанного на потоковом сокете?
5. Алгоритм установления связи между клиентом и сервером для взаимодействия на основе потокового сокета.
6. Алгоритм прослушивания портов на локальном компьютере.
7. Как завершить соединение между клиентом и сервером?
8. Какие сокеты участвуют при взаимодействии приложений?
9. Как осуществляется прием и отправка данных между клиентом и сервером?