Сокеты

Сокет

- Сокет – программный интерфейс для обеспечения обмена данными между процессами.
- Впервые socket API появилась в BSD Unix. Описан в POSIX
- В программе сокет идентифицируется *дескриптором* – переменной типа int.

Аттрибуты сокета

- С каждым сокет связываются три атрибута: *домен*, *тип* и *протокол*. Эти атрибуты задаются при создании сокета и остаются неизменными на протяжении всего времени его существования

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

Домен сокета

- AF_UNIX – межпроцессное взаимодействие
- AF_INET – TCP/IP, IPv4
- AF_INET6 – IPv6
- AF_IPX – IPX
- ….

Тип сокета

- **SOCK_STREAM**. Передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных (**TCP**)
- **SOCK_DGRAM**. Передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным (**UDP**)
- **SOCK_RAW**. Этот тип присваивается низкоуровневым сокетам. (IP, ICMP, ARP…)

Протокол сокета

- 0 – протокол по умолчанию
- Остальные варианты – RTFM :)

Схема работы сетевого приложения

- Создание сокета. Выбор домена и типа
- Привязка к определенному адресу и порту. Если пропустить, то произвольный порт.
- Подключение, если пропустить то создается канал без установления соединения.
- Прием или передача
- Разрыв соединения
- Закрытие сокета

Привязка сокета

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *addr, int addrlen);

struct sockaddr {
    unsigned short sa_family; // Семейство адресов,
    AF_xxx
    char sa_data[14]; // 14 байтов для хранения адреса
};
```

Используют **sockaddr_XX** (sockaddr_un, sockaddr_in)

Sockaddr_in

```
struct sockaddr_in {
    short int sin_family; // Семейство адресов
    unsigned short int sin_port; // Номер порта
    struct in_addr sin_addr; // IP-адрес
    unsigned char sin_zero[8]; // "Дополнение" до
                    размера структуры sockaddr
};

struct in_addr {
    unsigned long s_addr;
};
```

Хранение данных

- Существует два порядка хранения байтов в слове и двойном слове
- *порядок хоста* (host byte order)
- *сетевой порядок* (network byte order)

- **htons** (Host TO Network Short)
- **htonl** (Host TO Network Long)
- Обратно: **ntohs** и **ntohl**.

Хранение IP адреса в sockaddr

Для преобразования IP адреса («127.0.0.1») в in_addr используется

int **inet_aton**(const char *cp, struct in_addr *in_p);

Обратно:

char ***inet_ntoa**(struct in_addr in);

```
#define SERVER_ADDR     "127.0.0.1«
inet_aton(SERVER_ADDR, &dest.sin_addr.s_addr)
```

Установка соединения (клиент)

- #include <sys/types.h>
- #include <sys/socket.h>
- int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

- Делать bind не обязательно!

Установка соединения (сервер)

bind:

- Если в качестве адреса **INADDR_ANY** – любой интерфейс
- Порт = 0 – система выберет сама

int listen(int sockfd, int backlog);

Переводит сокет в режим ожидания запросов от клиентов. Backlog – размер очереди запросов.

Установка соединения (сервер)

int accept(int sockfd, void *addr, int *addrlen);

Функция **accept** создаёт для общения с клиентом *новый* сокет и возвращает его дескриптор.

addr – клиент. Можно сделать NULL, если не интересует.

Обмен данными

- int send(int sockfd, const void *msg, int len, int flags);

- Функция **send** возвращает число байтов, которое на самом деле было отправлено (или -1 в случае ошибки) !

Обмен данными

```
int sendall(int s, char *buf, int len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = send(s, buf+total, len-total, flags);
        if(n == -1)
            break;
        total += n;
    }
    return (n==-1 ? -1 : total);
}
```

Обмен данными

int recv(int sockfd, void *buf, int len, int flags);

Функция **recv** возвращает количество прочитанных байтов, которое может быть меньше размера буфера.

Закрытие сокета

```
#include <unistd.h>
int close(int fd);
```

Не забываем!

Диагностика ошибок

- #include <errno.h>
- errno – глобальная переменная, хранящая код последней ошибки.

```
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
{
    perror("Socket");
    exit(errno);
}
```

Простой клиент (инициализация)

```
    int sock;
    struct sockaddr_in addr;

    if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
       perror("socket");
       exit(errno);
    }

    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(3425); // порт...
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
//     #define SERVER_ADDR     "127.0.0.1"
//     inet_aton(SERVER_ADDR, &addr.sin_addr.s_addr);

    if(connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
       perror("connect");
       exit(errno);
    }
```

Простой клиент (прием-передача)

```
send(sock, message, sizeof(message), 0);
recv(sock, buf, sizeof(message), 0);

printf(buf);
close(sock);
```

Сервер - инициализация

```
int sock, listener;
   struct sockaddr_in addr;
   char buf[1024];
   int bytes_read;

   listener = socket(AF_INET, SOCK_STREAM, 0);
   if(listener < 0)
   {
       perror("socket");
       exit(errno);
   }

   addr.sin_family = AF_INET;
   addr.sin_port = htons(3425);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   if(bind(listener, (struct sockaddr *)&addr, sizeof(addr)) < 0)
   {
       perror("bind");
       exit(errno);
   }
```

Сервер - работа с клиентами

```
listen(listener, 1);
while(1)
{
    sock = accept(listener, NULL, NULL);
    if(sock < 0)
    {
        perror("accept");
        exit(errno);
    }

    while(1)
    {
        bytes_read = recv(sock, buf, 1024, 0);
        if(bytes_read <= 0) break;
        send(sock, buf, bytes_read, 0);
    }

    close(sock);
}
```

Обмен датаграммами (UDP)

int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

БЕЗ УСТАНОВКИ СОЕДИНЕНИЯ!

to, tolen, from, fromlen – адрес и его длина.