Лабораторная работа U12

Взаимодействие по компьютерным сетям, сокеты

Для взаимодействия компьютеров используются т.н. компьютерные сети, на сегодняшний день – с протоколом TCP/IP. На каждое сетевое взаимодействие есть свой протокол обмена, есть рассчитанные на передачу двоичных данных, а есть рассчитанные на передачу обычного текста (ASCII).

Сокет (socket) - это конечная точка сетевых коммуникаций, используется для сокрытия тонкости сетевых протоколов от программиста. В программе сокет идентифицируется *дескриптором* - это переменная типа **int**, аналогично указателю на открытый файл. Программа получает дескриптор от операционной системы при создании сокета, а затем передаёт его сервисам socket API для указания сокета, над которым необходимо выполнить то или иное действие.

Есть сокеты блокирующие и не блокирующие.

Основные системные вызовы для работы с сокетами.

Создание сокета: функция **socket**, имеющая следующий прототип

int socket(int domain, int type, int protocol);

где domain это тип сокета, для IPv4 используется AF_INET

а type – для AF_INET может принимать значения **SOCK_STREAM** (протокол TCP), **SOCK_DGRAM** (UDP), **SOCK_RAW** (для низкоуровневого формирования пакетов)

protocol зависит от параметра 1 и 2. Для IPv4 и TCP(UDP) достаточно указать 0.

Закрытие сокета

```
int close(int fd);
```

…или отключение передачи данных в одном направлении:

int shutdown(int sockfd, int how);

Структура для определения сокета

```
#include <netinet/in.h>
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
```

Пример:

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;        ---  IPv4
addr.sin_port = htons(4321);   ---  порт N 4321
addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

где INADDR_ANY – любой доступный IP адрес компьютера, или INADDR_LOOPBACK – 127.0.0.1

Установка соединения используется connect:

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

пример: connect(sock, (struct sockaddr *)&addr, sizeof(addr));

Привязка сокета к компьютеру адресу/порту компьютера

int bind(int sockfd, struct sockaddr *addr, int addrlen);

Функция возвращает значение <0 если произошла ошибка, например порт уже используется другой программой (ранее запущенной копией)

пример вызова:

bind(listener, (struct sockaddr *)&addr, sizeof(addr))

перевод сокета в режим ожидания

int listen(int sockfd, int backlog);

где backlog – это количество запросов на установку соединения которые могут одновременно находится в очереди. Если значение =5, и в очереди 5 запросов, то 6 и последующие попытки установить соединение будут отвергнуты

ожидание и приём соединения

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

Функция **accept** создаёт для общения с клиентом *новый* сокет и возвращает его дескриптор. Где **sockfd** задаёт слушающий сокет. В структуру, на которую ссылается **addr**, записывается адрес сокета клиента, который установил соединение с сервером. В переменную, адресуемую указателем **addrlen**, изначально записывается размер структуры; Если адрес клиента не требуется, надо передать NULL в качестве второго и третьего параметров.

Обмен данными - функции **send** и **recv**.

В Unix для работы с сокетами можно использовать также файловые функции **read** и **write**, но они обладают меньшими возможностями, а кроме того не будут работать на других платформах (например, под Windows)

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).

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

По аналогии с **send** функция **recv** возвращает количество прочитанных байтов, которое может быть меньше размера буфера. Если **recv** вернула 0, это означает, что соединение было разорвано.

План программ серверов

1. Создается сокет
2. Заполняется структура sockaddr где указывается ip адрес на который надо привязать сокет(или на все доступные на компьютере адреса) и порт
3. Привязывается к IP адресу / порту
4. Сокет переводится в режим ожидания входящих соединений
5. В цикле:
    a. Вызывается команда ожидания входящих соединений
    b. Создаётся дубликат процесса. Родительский переходит в начало цикла и ожидает следующего соединения
    c. Процесс-потомок – осуществляет обмен данными

6. Закрытие сокета

План программы клиента

1. Создаётся сокет
2. Заполняется структура sockaddr где указывается IP адрес и порт к которым подключаться
3. Устанавливается соединение с удалённым сервером
4. Происходит обмен данными
5. Закрытие сокета

Необходимые библиотеки для сокетов

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

Для компилирования используется команда gcc –o programa program.c , Где programa – имя исполняемого файла который должен получиться, program.c – имя файла с исходным кодом программа

Задание:

1. Написать программу-сервер реализующую:
    a. Открывается TCP порт, номер любой выше 1000 чтобы не пересечься с уже существующим ПО. Номер порта – можно зашить в коде.
    b. При соединении клиенту выдаётся какой-либо приветственный текст
    c. Всё что клиент присылает, выводится на экран терминала с запущенной программой-сервером (альтернативный вариант – в системный журнал) и возвращается обратно пользователю (по сети) без какой-либо дополнительной обработки (т.н. «эхо»)
2. С помощью команды telnet проверить её работу:

    telnet localhost XXXX

    где XXXX номер использованного порта
3. Написать программу-клиент, подключающуюся к программе с п1 на локальной машине и передающую зашитый в коде текст. Полученные данные от сервера напечатать на экране.