Системное программирование

Inter-process communication

IPC

"The last thing one knows in constructing a work is what to put first". -- Blaise Pascal

«Лишь в конце работы мы обычно узнаем, с чего ее нужно было начать». – Блез Паскаль

Сокеты и удаленное межпроцессное взаимодействие

Лабораторная работа #2

Сергей Станкевич, Минск

ЛАБОРАТОРНАЯ РАБОТА №10.

Клиент-серверное сокетное соединение в сети.

ЦЕЛЬ. Изучить механизм сокетного сетевого взаимодействия в LINUX.

Норма времени выполнения: 2 академических часа.
Оценка работы: 3 зачетных единицы.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

Понятие – сокет

Удаленное межпроцессное взаимодействие осуществляется посредством сокетов. *Сокет* – это комплексное понятие. Условно можно назвать "точкой соединения процессов", или «портом». Это универсальный способ взаимодействия процессов. Сокеты могут использоваться как для *локального* взаимодействия *родственных* процессов, так и для обмена данными между *удаленными разнородными* системами.

Сокеты – это *специальный файл*, один из семи типов файлов в UNIX (OS BSD). С сокетами можно проводить такие-же как с обычными файлами. В отличие от обычных файлов, сокеты представляют собой *виртуальный объект*, который существует, пока на него ссылается хотя бы один из процессов.

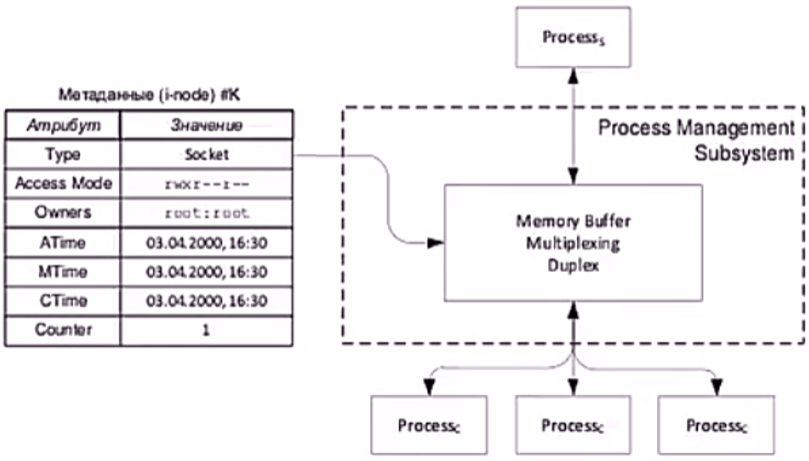


Рисунок 1 - Структура файла-сокета

Через этот специальный файл происходит взаимодействие между локальными или удаленными процессами.

В программах сокеты фигурируют в виде *файловых дескрипторов*, над которыми, во многих случаях, можно осуществлять обычные операции чтения-записи (read(), write() и т.п.). Но *набор действий*, подготавливающих файловый дескриптор к использованию, *несколько сложнее*, чем в случае с обычными файлами.

Коммуникационный домен

Итак, являются коммуникационным интерфейсом взаимодействующих процессов. Взаимодействие между процессами должно быть унифицировано, независимо от того, выполняются ли они на одном компьютере или на разных хостах сети.

При взаимодействии посредством сокетов процессы рассматриваются по схеме "**клиент - сервер**". Для установки взаимодействия между процессами посредствам сокета необходимо установить соединение между процессами, установить **коммуникационный канал**.

В общем случае каждый *коммуникационный канал* определяется двумя узлами – источником и получателем данных.

Создание сокета не означает создания *коммуникационного узла*. Для однозначной идентификации сокета его необходимо позиционировать в пространстве имен конкретного *коммуникационного домена*.

В этой связи было введено понятие **коммуникационный домен** (communication domain). *Коммуникационный домен* описывает набор определенных характеристик взаимодействия, различные схемы адресации объектов, их расположение, протоколы передачи данных и т. д.

Для обозначения коммуникационного узла, обеспечивающего прием и передачу данных для объекта (процесса), был предложен специальный объект – socket. Сокеты создаются в рамках определенного коммуникационного домена, подобно тому, как файлы создаются в рамках файловой системы. Сокеты имеют соответствующий интерфейс доступа в файловой системе UNIX, и так же, как обычные файлы, адресуются некоторым целым числом – дескриптором. Однако в отличие от обычных файлов, представляют собой *виртуальный объект*, который существует, пока на него ссылается хотя бы один из процессов.

Во-вторых, коммуникационные характеристики взаимодействия должны быть доступны процессам в некоторой унифицированной форме. Другими словами, приложение должно иметь возможность затребовать определенный тип связи, например, основанный на виртуальном канале (virtual circuit) или датаграммах (datagram), причем эти типы должны быть согласованы для всех коммуникационных доменов.

Полный набор коммуникационных характеристик включает:

- Упорядоченную доставку данных
- Отсутствие дублирования данных
- Надежную доставку данных
- Сохранение границ сообщений
- Поддержку передачи экстренных сообщений
- Предварительное установление соединения

Например, именованные каналы, обеспечивают только первые три характеристики.

Взаимодействие с предварительным установлением соединения предполагает создание виртуального канала между источником и получателем данных. Это избавляет от необходимости идентифицировать передающую сторону в каждом пакете данных. Идентификация происходит на начальном этапе установления связи и затем сохраняется для всех пакетов, принадлежащих данному виртуальному каналу.

Наконец, для того чтобы независимые процессы имели возможность взаимодействовать друг с другом, для сокетов должно быть определено пространство имен. Имя сокета имеет смысл только в рамках коммуникационного домена, в котором он создан. Если для System V используются ключи, то имена сокетов представлены адресами.

Типы сокетов

Сокеты – это универсальный способ взаимодействия процессов. Классификацию сокетов можно провести по двум критериям: *семейству протоколов* и *способу взаимодействия*.

Ядро Linux поддерживает несколько десятков *семейств протоколов* для сокетов. Некоторые из них встречаются повсеместно, другие – очень редко. Наиболее распространенные это:

- **Unix-сокеты** для *локального* взаимодействия процессов.
- **Интернет-сокеты** для *удаленного* взаимодействия процессов.

По *способу взаимодействия* сокеты делятся на несколько типов:

- **Сокет дейтограмм** (datagram socket), через который осуществляется теоретически ненадежная, несвязная передача пакетов.
- **Сокет потока** (stream socket), через который осуществляется надежная передача потока байтов без сохранения границ сообщений. Этот тип сокетов поддерживает передачу экстренных данных.
- **Сокет пакетов** (packet socket), через который осуществляется надежная последовательная передача данных без дублирования с предварительным установлением связи. При этом сохраняются границы сообщений.
- **Сокет низкого уровня** (raw socket), через который осуществляется непосредственный доступ к коммуникационному протоколу.

Мы рассмотрим два типа сокетов: потоковые и дейтаграммные.

Потоковые сокеты служат для организации надежной передачи данных в сети через протоколы, обеспечивающие контроль целостности доставки и приема информации. Наиболее распространенный протокол такого типа – TCP (Transmission Control Protocol).

Дейтаграммные сокеты, напротив, ориентированы не на надежную, а на быструю передачу информации. В сети такие данные чаще всего передаются по протоколу UDP (User Datagram Protocol), который, в отличие от TCP, не контролирует целостность доставки информации и не создает предварительных соединений между клиентом и сервером.

Создание сокетов

Для работы с сокетами предусмотрены специальные *системные вызов*, которые объявлены в заголовочном файле **sys/socket.h**.
Для создания сокетов предназначен системный вызов socket():

```
int socket (int PF, int SOCK_TYPE, int PROTOCOL);
```

Аргументы:
PF – это семейство протоколов сокета;
SOCK_TYPE – способ взаимодействия;
PROTOCOL – необходим для задания конкретного протокола передачи данных. Если здесь указать 0, то выбор протокола произойдет автоматически.

Данный системный вызов при успешном завершении возвращает дескриптор сокета. Если произошла ошибка, то возвращается -1.

Семейство протоколов определяется одной из символических констант, которые объявлены в файле bits/socket.h. Вот некоторые семейства протоколов:

PF_LOCAL — локальные сокеты (Unix-сокеты);
PF_UNIX — синоним PF_LOCAL (для совместимости с BSD-системами);
PF_FILE — нестандартный синоним PF_LOCAL;
PF_INET — интернет-сокеты, основанные на IP версии 4 (протокол Internet TCP/IP);
PF_INET6 — интернет-сокеты, основанные на IP версии 6;
PF_BLUETOOTH — bluetooth-сокеты.

> Префикс PF (protocol family) – семейство протоколов.
> Допустимыми также являются названия с префиксом AF (от address family — семейство адресов), например, AF_UNIX, AF_INET, AF_NS т. д.

Заголовочный файл bits/socket.h не предназначен для непосредственного включения в пользовательские программы. Для доступа к константам семейств протоколов используйте файл socket.h.

Способ взаимодействия также определяется одной из констант, находящихся в заголовочном файле bits/socket.h. Нам понадобятся только две константы:

SOCK_STREAM — потоковые сокеты;
SOCK_DGRAM — дейтаграммные сокеты.

Создание сокета не означает создания *коммуникационного узла*. Для однозначной идентификации сокета его необходимо позиционировать в пространстве имен данного коммуникационного домена. В общем случае каждый *коммуникационный канал* определяется *двумя узлами – источником* и *получателем* данных.

Коммуникационный канал может характеризоваться тремя параметрами:

1. Коммуникационным протоколом
2. Адресом (локальным или удаленным)
3. Процессом (локальным или удаленным)

Как правило, адрес определяет операционную систему (или хост сети), а процесс – конкретное приложение, получающее или передающее данные. Однако конкретные значения и формат этих параметров определяются коммуникационным доменом.

Установление соединения между процессами

Поскольку при создании сокета указывается только один параметр – коммуникационный протокол, прежде чем передача данных между взаимодействующими процессами станет возможной необходимо указать четыре дополнительных параметра для коммуникационного канала. *Процедура установки* этих параметров существенным образом зависит от следующих факторов:

- типа создаваемого канала (определяется типом используемого сокета);
- коммуникационного протокола.

Очевидно, что взаимодействующие стороны должны делать это согласованно, используя либо заранее определенные адреса, либо договариваясь о них в процессе установления связи. *Установление соединения* между процессами с использованием сокетов проводится двумя способами:

- С предварительным установлением соединения
- Без предварительного установления соединения

Назначение адреса: bind()

Чтобы сервер и клиент могли взаимодействовать, сокету нужно назначить адрес (имя). В зависимости от типа сокета адресом может являться:

- имя файла (локальное взаимодействие);
- сетевой адрес и порт (удаленное взаимодействие).

Адрес сокета назначается на стороне сервера. Клиентский процесс может использовать этот адрес для подсоединения к серверу или для передачи данных.

Назначение адреса сокету осуществляется системным вызовом bind():

```
int bind (int FD, const struct sockaddr * ADDRESS, socklen_t LEN);
```

Аргументы:
FD – это дескриптор сокета.
Структура типа sockaddr, расположенная по адресу ADDRESS задает адресное пространство и сам адрес.
LEN указывает размер адресной структуры во втором аргументе.

Типы структур, указываемых через механизм приведения типов языка Си, во втором аргументе bind():

- struct sockaddr_un – для локальных сокетов;
- struct sockaddr_in – для интернет-сокетов.

В свою очередь структура sockaddr_un содержит два поля, которые нас интересуют:

- sun_family – семейство адресов;
- sun_path – имя (путь) файла сокета.

Размер структуры sockaddr_un можно вычислить при помощи функции sizeof() или специального макроса SUN_LEN().

В качестве sun_family при локальном взаимодействии на основе Unix-сокетов обычно устанавливается константа AF_LOCAL или ее синоним AF_UNIX. Поле sun_path – это обычная строка, содержащая путь к файлу сокета. Поле sun_path структуры sockaddr_un – это не просто указатель, а статический массив символов. Запись данных в этот элемент выполняйте функциями наподобие strcpy() или sprintf().

В структуре sockaddr_in обычно устанавливаются следующие поля:

- sin_family – семейство адресов;
- sin_addr – адрес сокета;
- sin_port – порт сокета.

Размер sockaddr_in вычисляется только через sizeof().

Поле sin_addr – это приведенный к целому числу IP-адрес серверного узла. Обычно адреса IP записываются в виде "четверки с точками" (например, 192.168.0.1). Кроме того, в Интернете распространена система доменных имен (DNS, Domain Name System), которая сопоставляет IP-адресам удобочитаемые имена (домены). Для преобразования "четверки с точками" или доменного имени в числовой адрес существует библиотечная функция gethostbyname(), которая объявлена в заголовочном файле netdb.h следующим образом:

```
struct hostent * gethostbyname (const char * NAME);
```

NAME – это доменное имя (адрес) в виде "четверки с точками".

Адрес узла обычно находится в элементе host->h_addr_list[0] структуры hostent.

> Не следует путать доменные имена и адреса URL (Uniform Resource Locator). Адреса URL служат для определения местонахождения конкретных ресурсов в сети Интернет, а домены — это просто имена IP-адресов в системе DNS.
> Например, http://www.bhv.ru/ — это URL-адрес главной страницы сайта издательства "БХВ-Петербург". Однако доменным именем сервера будет просто **bhv.ru**.

Дело в том, что сервер может одновременно обмениваться данными с несколькими клиентами. Кроме того, на одном узле может работать несколько

различных серверов. Для решения данной проблемы используют порты. Порт – это число, идентифицирующее сеанс межпроцессного взаимодействия.

В поле sin_port структуры sockaddr_in содержится 16-разрядный номер порта. Дело в том, что порядок записи байтов и битов на конкретном компьютере может отличаться от той, что принята в сети общего пользования. Поэтому для преобразования локального числа в 16-разрядный "сетевой" формат необходима специальная функция htons().

Взаимодействие сокетов с предварительным установлением соединения

В клиент-серверном взаимодействии процессов с предварительным соединением сокетов проводятся следующие операции.

Основные операции процесса-сервера:

- Создание сокета
- Назначение сокету адреса
- Прослушивание сокета
- Принятие запроса
- Закрытие (удаление) сокета

Основные операции процесса-клиента:

- Создание сокета
- Соединение с сервером
- Отсоединение от сервера

Проиллюстрируем на рисунке 2 взаимодействие между процессами при виртуальном коммуникационном канале с *предварительным установлением связи*.

Далее рассмотрим необходимые *системные вызовы* для установки такого соединения.

Соединение сокетов: connect()

При использовании *потоковых* сокетов между взаимодействующими процессами должно сначала установиться соединение. Для этого клиент вызывает системный вызов connect():

```
int connect (int FD, const struct sockaddr * SADDR, socklen_t LEN);
```

FD – дескриптор сокета;
SADDR – указатель на структуру, содержащую сведения об адресе сервера;
LEN – размер адресной структуры.

При успешном выполнении connect() возвращает 0, в случае ошибки -1.

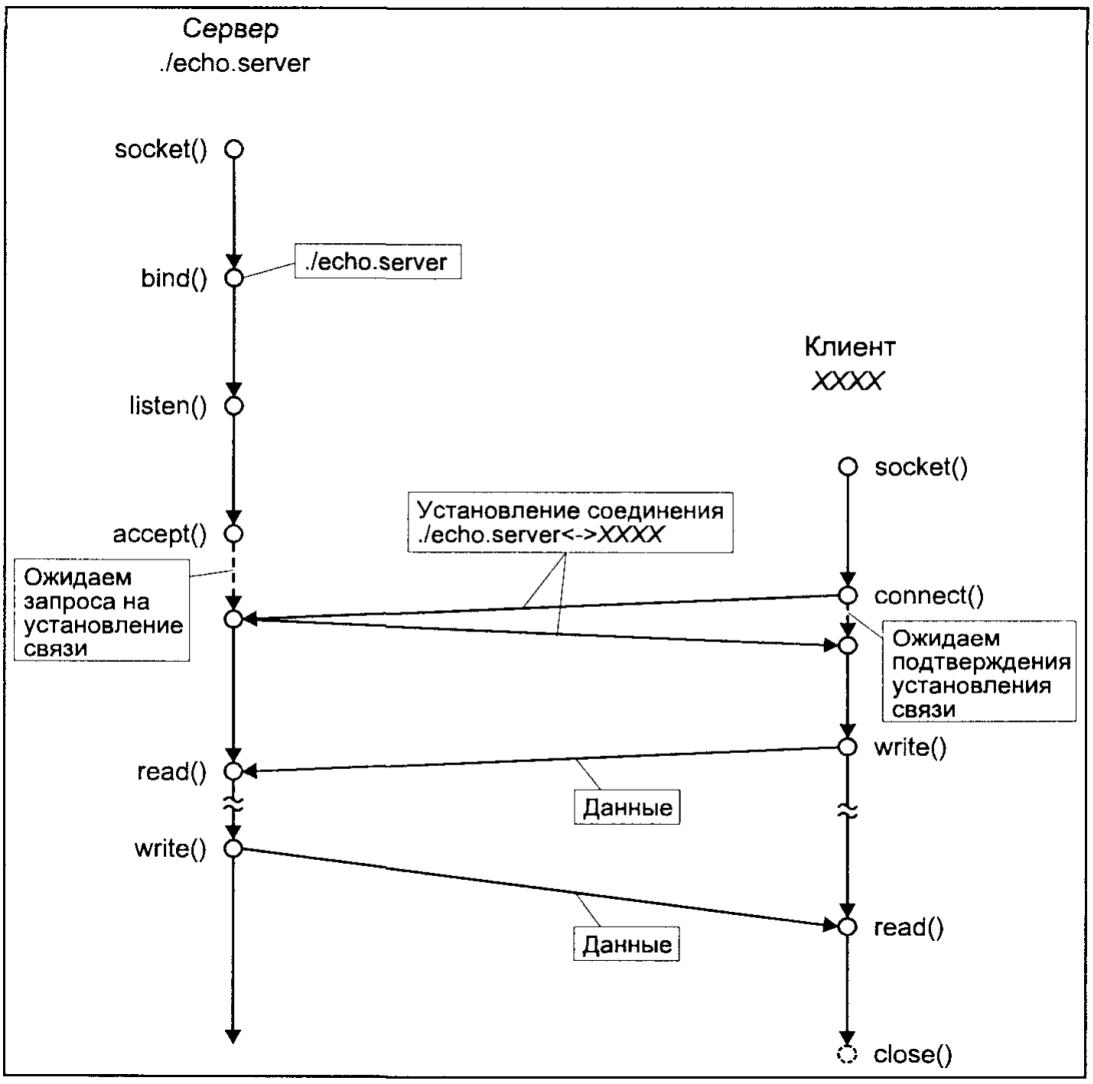


Рисунок 2 (3.21.) – Взаимодействие между процессами при создании виртуального канала (с предварительным установлением соединения)

Прослушивание сокета: listen()

При взаимодействии через *потоковые* сокеты сервер должен включить прослушивание, т. е. перейти в режим ожидания запросов на подключение. Это делается при помощи системного вызова listen():

```
int listen (int FD, int QUEUE_LEN);
```

FD – дескриптор сокета.
QUEUE_LEN – определяет максимальный размер очереди запросов на подключение.

При успешном выполнении listen() возвращает 0, в случае ошибки -1.

Принятие запроса на подключение: accept()

Системный вызов listen() блокирует сервер до тех пор, пока какой-нибудь клиент не выдаст запрос на подключение. Как только запрос поступил, сервер "просыпается". Если есть возможность обслужить запрос, то сервер вызывает системный вызов accept():

```
int accept (int FD, struct sockaddr * ADDRESS, socklen_t * LENP);
```

FD — это дескриптор сокета.
По адресу ADDRESS расположена структура, в которую помещаются адресные данные созданного соединения (в частности – порт). Если вам не нужны эти данные, можете просто указать NULL.
LENP – адрес переменной, в которую помещается размер адресной структуры.

При успешном выполнении accept() возвращает *дескриптор нового сокета*, который будет обслуживать созданное соединение. В случае ошибки возвращается -1.

Взаимодействие сокетов без предварительного установлением соединения

Для быстрой, но менее надёжной передачи данных применяются *дейтаграммные* сокеты. При работе с потоковыми сокетами применяются обычные операции ввода-вывода. Однако в случае дейтаграммных сокетов такой подход невозможен. Дело в том, что дейтаграммные сокеты не предназначены для установки соединений посредством системных вызовов connect(), listen() и accept(). Для передачи информации без использования соединений нужны системные вызовы, которые посылают и принимают данные *через непосредственные адреса*.

Для дейтаграммных сокетов на стороне клиента сокет предусмотрен системный вызов sendto(), а на стороне сервера – recvfrom(). Эти системные вызовы объявлены в заголовочном файле sys/socket.h.

Для *передачи* данных на сервер через дейтаграммный сокет предусмотрен системный вызов sendto():

```
ssize_t sendto (int FD, const void * BUFFER,
        size_t BUF_SIZE, int FLAGS,
        const struct sockaddr * SADDR,
        socklen_t * LEN);
```

FLAGS позволяет передавать дополнительные флаги; если таковых не имеется, пишите 0.
Через SADDR передается адресная структура сервера, на который посылаются данные.
LEN — это размер этой адресной структуры.

Возвращаемое значение и первые три аргумента полностью идентичны тем, что используются в системном вызове write().

На *серверной* стороне данные *принимаются* при помощи системного вызова recvfrom():

```
ssize_t recvfrom ( int FD,
        void * BUFFER,
        size_t BUF_SIZE,
        int FLAGS,
        struct sockaddr * CADDR,
        socklen_t * LENP);
```

FLAGS позволяет передавать дополнительные флаги.
Через CADDR сервер может получить адресную структуру клиента,
а через LENP – размер этой структуры.

Этот системный вызов, кроме прочего, позволяет получить адрес отправителя, что важно в тех случаях, когда клиент ожидает ответ от сервера. Возвращаемое значение и первые три аргумента идентичны тем, что используются в системном вызове read().

На рисунке 3 представлено взаимодействие между процессами при виртуальном коммуникационном канале с без *предварительного установления соединения*.

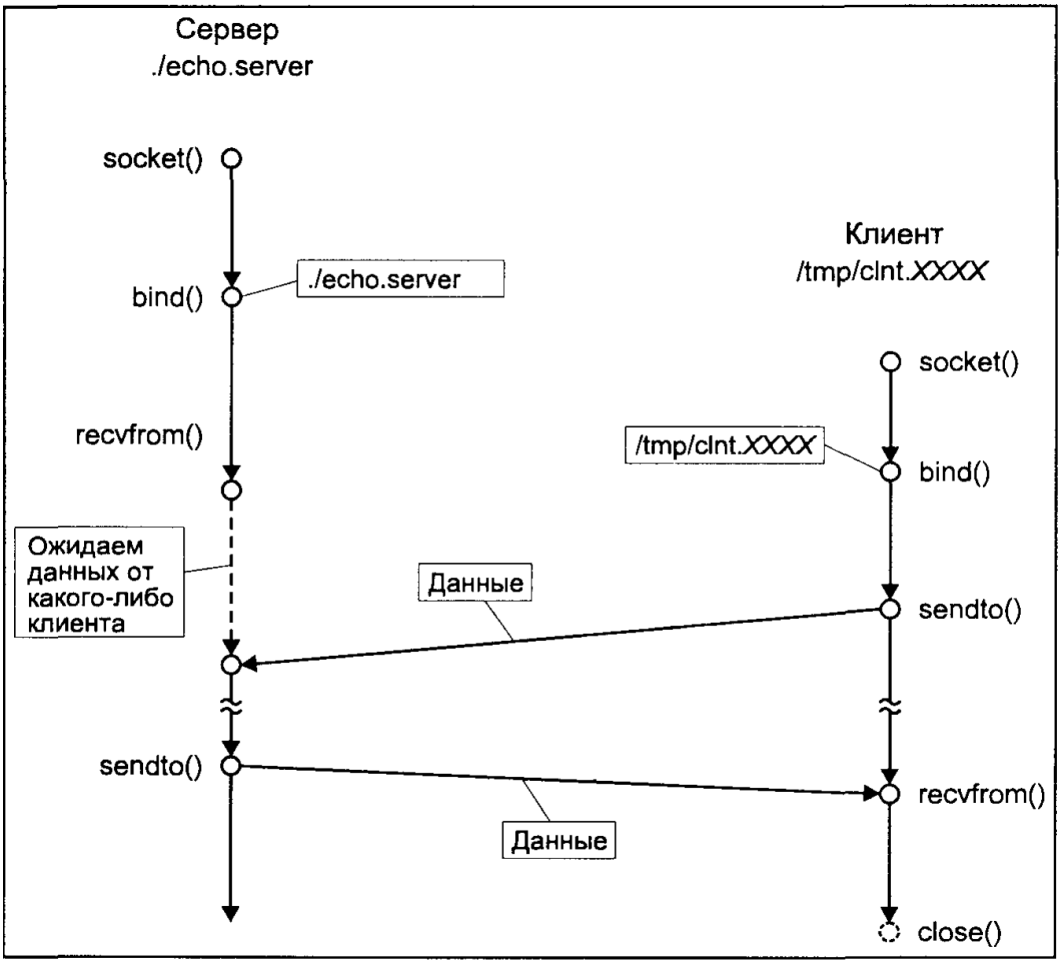


Рисунок 3 (3.22.) – Взаимодействие между процессами, основанное на датаграммах (*без предварительного установления соединения*)

Best of LUCK with it, and remember to HAVE FUN while you're learning :)

УПРАЖНЕНИЯ

Упражнение 1

Создание локального сокета и назначения ему локального адреса.

Пример №1. Создание локального сокета и назначения ему локального адреса.

Программа **socket1.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <sys/un.h>

int main (int argc, char ** argv)
{
        int sock;
        struct sockaddr_un saddr;

        if (argc < 2) {
        fprintf (stderr, "Too few arguments\n");
        return 1;
        }

        sock = socket (PF_UNIX, SOCK_STREAM, 0);
        if (sock == -1) { fprintf (stderr,
        "socket() error\n"); return 1;
        }

        saddr.sun_family = AF_UNIX;
        strcpy (saddr.sun_path, argv[1]);

        if (bind (sock, (struct sockaddr *)
        &saddr, SUN_LEN (&saddr)) == -1) {
        fprintf (stderr, "bind() error\n");
        return 1;
        }
        fprintf (stderr, "Press <Enter> to con-
        tinue...");
        fgetc (stdin);

        close (sock);
        unlink (argv[1]);
        return 0;
}
```

Проверяем:

```
$ gcc -o socket1 socket1.c
$ ./socket1 mysocket
Press <Enter> to continue...
```

Откроем другое терминальное окно и посмотрим на наш сокет:

```
$ ls -l mysocket
srwxr-xr-x  1  nn  nn  0  2011-05-11  10:18
mysocket
```

Обратите внимание, что в расширенном выводе программы ls сокет обозначается символом s. После нажатия клавиши <Enter> в первом терминале сокет исчезнет.

Какой консольной командой и каким системным вызовом в программах удаляется сокет из системы?

Пример №2. Чтение главной страницы из указанного Web-сервера.

Программа **getwwwpage.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>

#define BUF_LEN         4096
#define HTTP_PORT       80
int main (int argc, char ** argv)
{
        int sock, count;
        char * buf;
        struct hostent * host;
        struct sockaddr_in addr;

        if (argc < 2) { fprintf (stderr, "Too few ar-
        guments\n"); return 1;
        }
        buf = (char *) malloc (BUF_LEN);
        if (buf == NULL) {
        fprintf (stderr, "malloc() error\n");
        return 1;
        }

        sock = socket (PF_INET, SOCK_STREAM, 0);
        if (sock == -1) { fprintf (stderr, "socket()
        error\n"); return 1;
        }

        addr.sin_family = AF_INET;

        host = gethostbyname (argv[1]);
        if (host == NULL) {
        fprintf (stderr, "Unknown server\n");
        return 1;
        }

        addr.sin_addr = * (struct in_addr*) host-
        >h_addr_list[0];

        addr.sin_port = htons (HTTP_PORT);

        if (connect (sock, (struct sockaddr*)
        &addr, sizeof (addr)) == -1) { fprintf
        (stderr, "connect() error\n"); return 1;
        }

        strcpy (buf, "GET /\n");
        write (sock, buf, strlen (buf));

        while ((count = read (sock, buf,
        BUF_LEN)) > 0)
                write (1, buf, count);

        close (sock);
        free (buf);
        return 0;
}
```

Нужно подключится к Интернету.

```
$ gcc -o getwwwpage getwwwpage.c
$ ./getwwwpage bhv.ru > index.html
```

Посмотрите содержимое файла index.html.

Упражнение 2

Межпроцессное взаимодействие с использованием локальных потоковых сокетов.

Создадим локальное клиент-серверное приложение.

Программа **socket2-server.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <sys/un.h>
#include <stdlib.h>

#define QUEUE_LENGTH     10
#define BUF_LEN          256
#define SOCK_NAME      "mysocket"

int main (void)
{
        int sock, client_sock;
        struct sockaddr_un saddr;
        char * buf;
        int count;

        sock = socket (PF_UNIX, SOCK_STREAM, 0);
        if (sock == -1) { fprintf (stderr, "socket()
        error\n"); return 1;
        }

        buf = (char *) malloc (BUF_LEN);
        if (buf == NULL) {
        fprintf (stderr, "malloc() error\n");
        return 1;
        }

        saddr.sun_family = AF_UNIX;
        strcpy (saddr.sun_path, SOCK_NAME);

        if (bind (sock, (struct sockaddr *) &saddr,
        SUN_LEN (&saddr)) == -1) { fprintf
        (stderr, "bind() error\n"); return 1;
        }
        if (listen (sock, QUEUE_LENGTH) == -1) {
        fprintf (stderr, "listen() error\n"); return
        0;
        }
```

Программа **socket2-client.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>

#define SOCK_NAME      "mysocket"

int main (int argc, char ** argv)
{
        int sock;
        struct sockaddr_un addr;

        if (argc < 2) { fprintf (stderr, "Too few ar-
        guments\n"); return 1;
        }
        sock = socket (PF_UNIX, SOCK_STREAM, 0);
        if (sock == -1) { fprintf (stderr, "socket()
        error\n"); return 1; }

        addr.sun_family = AF_UNIX;
        strcpy (addr.sun_path, SOCK_NAME);

        if (connect (sock, (struct sockaddr *)
        &addr, SUN_LEN (&addr)) == -1) {
        fprintf (stderr, "connect() error\n");
        return 1;
        }

        if (write (sock, argv[1], strlen (argv[1])) == -1)
        { fprintf (stderr, "write() error\n"); return 1;
        }

        close (sock);
        return 0;
}
```

// продолжение socket2-server.c

```
while (1) {
client_sock = accept (sock, NULL, NULL);

        if (client_sock == -1) { fprintf (stderr, "ac-
        cept() error\n");
        return 1;
        }

if ((count = read (client_sock, buf, BUF_LEN-1))
== -1) {
        fprintf (stderr, "read() error\n");
        return 1;
        }
```

```
        buf[count] = '\0';
        printf (">> %s\n", buf);

        close (client_sock);

        if (!strcmp (buf, "exit")) break;
        }

        free (buf);
        close (sock);
        unlink (SOCK_NAME);

        return 0;
}
```

Вскоре после запуска сервер переходит в режим прослушивания сокета:

```
$ gcc -o socket2-server socket2-server.c
$ ./socket2-server
```

Откроем другое терминальное окно и начнем передавать серверу запросы:

```
$ gcc -o socket2-client socket2-client.c
$ ./socket2-client Hello
$ ./socket2-client World
$ ./socket2-client Linux
```

Сервер будет "покорно" выводить сообщения до тех пор, пока клиент не пошлет строку "exit".

Упражнение 3

Прием и передача данных через дейтаграммные сокеты.

Программа **socket3-server.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <sys/un.h>
#include <stdlib.h>

#define QUEUE_LENGTH 10
#define BUF_LEN 256
#define SOCK_NAME "mysocket"
```

Программа **socket3-client.c**

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>



#define SOCK_NAME "mysocket"
```

```
// продолжение socket3-server.c

int main (void)
{
 int sock; int count;
 struct sockaddr_un saddr;
 char * buf;

 sock = socket (PF_UNIX, SOCK_DGRAM, 0);
 if (sock == -1) {
          fprintf (stderr, "socket() error\n");
          return 1;
 }

 buf = (char *) malloc (BUF_LEN);
 if (buf == NULL) {
          fprintf (stderr, "malloc() error\n");
          return 1;
 }

 saddr.sun_family = AF_UNIX;
 strcpy (saddr.sun_path, SOCK_NAME);

 if (bind (sock, (struct sockaddr *) &saddr,
SUN_LEN (&saddr)) == -1) {
          fprintf (stderr, "bind() error\n");
          return 1;
 }

 while (1) {
          if ((count = recvfrom (sock, buf,
          BUF_LEN-1, 0, NULL, NULL)) == -1) {
                    fprintf (stderr, "recvfrom() er-
                   ror\n");  return 1;
          }

          buf[count] = '\0';
          printf (">> %s\n", buf);

          if (!strcmp (buf, "exit")) break;
 }
          free (buf);
          close (sock);
          unlink (SOCK_NAME);
          return 0;
}
```

```
// продолжение socket3-client.c

int main (int argc, char ** argv)
{
 int sock;
 struct sockaddr_un addr;

 if (argc < 2) {
          fprintf (stderr, "Too few arguments\n");
          return 1;
 }

 sock = socket (PF_UNIX, SOCK_DGRAM, 0);
 if (sock == -1) {
          fprintf (stderr, "socket() error\n");
          return 1;
 }

 addr.sun_family = AF_UNIX;
 strcpy (addr.sun_path, SOCK_NAME);

 if (sendto (sock, argv[1], strlen (argv[1]), 0,
(struct sockaddr *) &addr, SUN_LEN (&addr)) ==
-1) {
          fprintf (stderr, "sendto() error\n");
          return 1;
 }

          close (sock);
          return 0;
}
```

Скомпилируйте и соберите программы. Запустите приложения и передайте с помощью них какие-нибудь данные. Объясните принцип работы приложения.

We hope you enjoy working with Linux!

ЗАДАНИЯ

Задание 1 – Потоковые сокеты

Напишите программы с клиент-серверной архитектурой по следующим требованиям:

1. Обменяться числами. Сервер возводит получаемое от клиента число в квадрат и возвращает ответ клиенту.
2. Создать два клиента на один и тот же сокет. Используют один и тот же сокет. Первый клиент получает результат счета от 1 до 10, второй – результат счета от 100 до 110 (подумайте, как распознать, какой клиент обратился к серверу).

> *Примечание*. Если количество студентов команды, выполняющих задание больше двух, то число процессов-клиентов должно соответствовать числу студентов команды.
>
> Если клиентов больше двух, то в этом случае процессы должны дополнительно выводить фамилии студентов, выполнивших задания. Третий клиент получает результат счета от 200 до 210. Каждому последующему клиенту сообщение увеличивается на 100.

3. Создать два клиента на один и тот же сервер. Сокеты должны быть разными (именованное соединение в каждом случае должно быть различным). Первый клиент получает результат счета от 1 до 10, второй – от 100 до 110.

> *Примечание*. Число процессов-клиентов и сокетов должно соответствовать числу студентов команды, выполняющей задание.
>
> Если клиентов больше двух, то в этом случае процессы должны дополнительно выводить фамилии студентов, выполнивших задания. Каждому последующему клиенту сообщение увеличивается на 100.

Задание 2 – Интернет сокеты

Напишите программу, которая читает главную страницу из какого-нибудь Web-сервера. Запуск программы должен производиться с консоли.

> **Варианты**, *простые числа* – подключиться к странице сайта БНТУ, ФИТР; *четные числа* – подключиться к странице сайта БНТУ, расписание занятий 3-го курса; *кратные трем* – подключиться к странице сайта БНТУ, абитуриент, поступление.

Задание 3 – Дейтаграммные сокеты

Напишите программу с клиент-серверной архитектурой для передачи данных через дейтаграммные сокеты. Количество клиентов должно соответствовать количеству членов вашей команды, но не менее двух. Семейства протоколов и способы взаимодействия сокетов-клиентов должны отличаться.

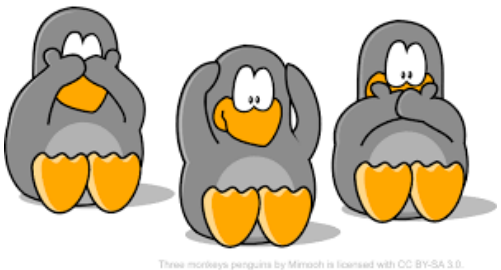
Запустите программы в разных терминалах и определите какие программы-клиенты получили данные. Объясните полученный результат.

Дополнительную информацию можно получить в книге Иванов Н. Н. «Программирование в Linux. Самоучитель» стр. 299 – 312.

«Easy things should be easy and hard things should be possible»
«Простые вещи должны быть простыми, а сложные вещи должны быть возможными»

Контрольные вопросы

1. Что такое удаленное межпроцессное взаимодействие?
2. Как переводится слово «socket» с английского на русском язык?
3. Что такое сокет? Как он представлен в операционной системе?
4. Какие операции можно производить над сокетом?
5. Какой способ взаимодействия процессов обеспечивает сокет?
6. Что такое архитектура приложений клиент/сервер, и в чем ее отличие от архитектуры взаимодействия процессов «источник» - «приёмник».
7. Что такое потоковые и дейтаграммные сокеты, для чего они используются?
8. Перечислите основные операции процесса-сервера.
9. Перечислите основные операции процесса-клиента.
10. Какие системные вызовы обеспечивают основные операции с сокетами?
11. Какой консольной командой и каким системным вызовом в программах удаляется сокет из системы?
12. Перечислите основные типы сокетов.
13. Что такое коммуникационный домен и семейство протоколов?



ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Литература

Шоттс У. «Командная строка Linux. Полное руководство.» — СПб.: Питер, 2017. — 480 с.: ил. — (Серия «Для профессионалов»).

Программирование в Linux. Самоучитель. — 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2012. — 400 с.: ил.

Робачевский А. М. Операционная система UNIX®. - СПб.: 2002. - 528 ил.

Справка

man (от англ. manual — руководство) — команда Unix, предназначенная для форматирования и вывода справочных страниц.

Интернет источники