Лекция 6

Сокеты

Межпроцессное взаимодействие

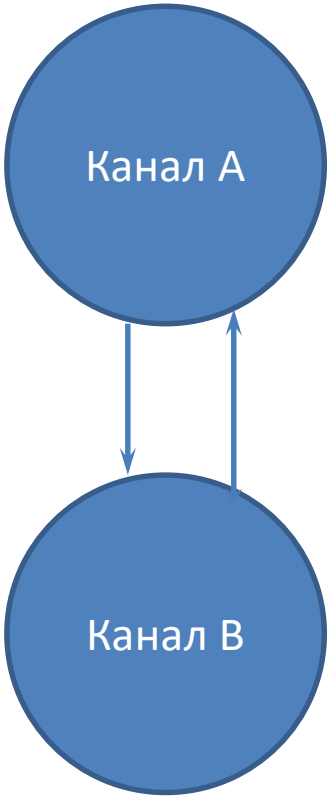
- Связанные процессы, совместно работающие над выполнением какой-нибудь задачи, зачастую нуждаются в обмене данными друг с другом и синхронизации своих действий. Такая связь называется **межпроцессным взаимодействием** (англ. inter-process communication, IPC).

Способы межпроцессного взаимодействия

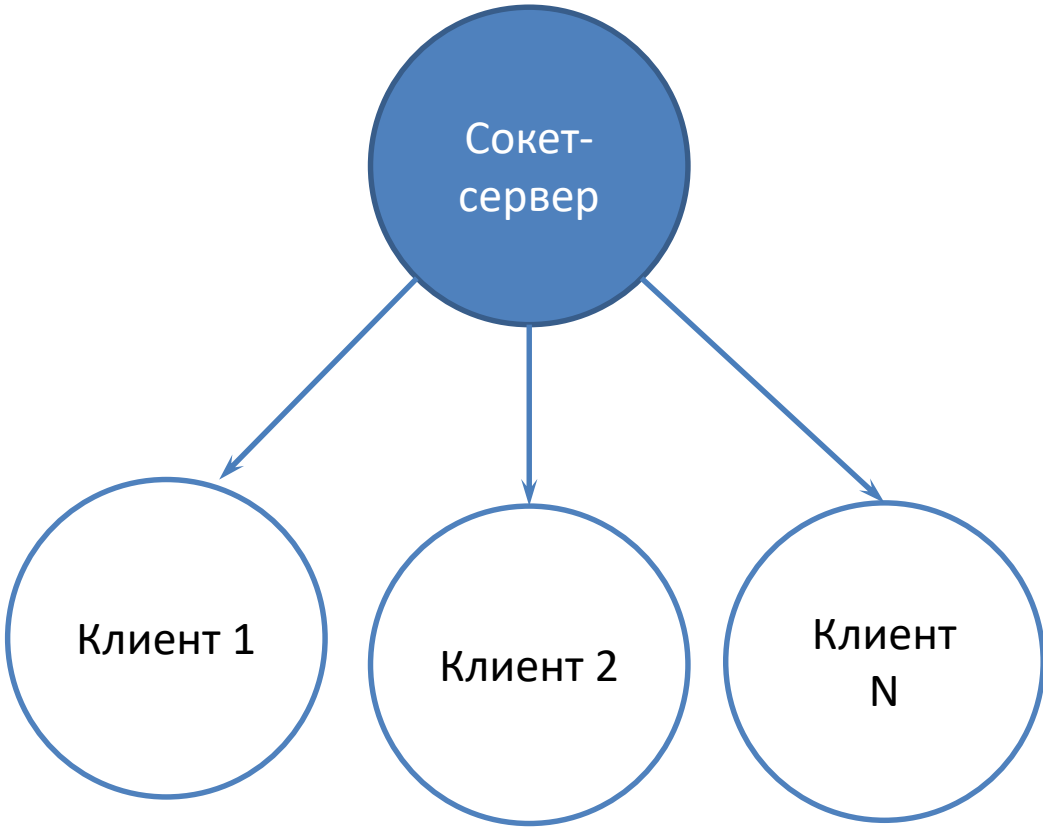
- **Файл**
- Сигнал
- **Сокет**
- Канал (конвейер)
- Именованный канал
- Неименованный канал
- Семафор
- Разделяемая память
- Проецируемый в память файл (mmap)
- Очередь сообщений (Message queue)

Каналы и сокеты

- Каналы



- Сокеты



Сокеты

- **Сокеты** – метод межпроцессного и сетевого взаимодействия как на одной вычислительной машине, так и в локальной сети.
- Сокет – конечная точка сетевых коммуникаций, являющаяся связующим звеном между одним или несколькими приложениями.
- Сокет, как и файл, ассоциируется **дескриптором**.

Структура сокета: домен

```
#include <sys/types.h>
#include <sys/socket.h>

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

Домен (*domain*) определяет пространство адресов, в котором располагается сокет, и множество протоколов, которые используются для передачи данных. Обычно, используются AF_UNIX (Unix) и AF_INET (Internet). **AF** обозначает **семейство адресов** (*address family*).

- **AF_UNIX** – передача данных происходит через файловую систему ввода/вывода.
- **AF_INET** – осуществляется сетевое взаимодействие.

Структура сокета: тип

```
#include <sys/types.h>
#include <sys/socket.h>

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

Тип (*type*) определяет способ передачи данных по сети. Чаще всего применяются:

SOCK_STREAM – передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных, при котором фрагменты отправленного блока не теряются, не переупорядочиваются и не дублируются. Используется чаще всего

SOCK_DGRAM – передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться в пути, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (multicasting) и широковещательная передача (broadcasting).

SOCK_RAW – этот тип присваивается низкоуровневым (т.н. "сырым") сокетам. Их отличие от обычных сокетов состоит в том, что с их помощью программа может взять на себя формирование некоторых заголовков, добавляемых к сообщению.

Структура сокета: тип

```
#include <sys/types.h>
#include <sys/socket.h>

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

Протокол (*protocol*) – часто протокол однозначно определяется по домену и типу сокета. В этом случае, можно передать 0, что соответствует протоколу по умолчанию.

Адреса

- Прежде чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене.
- Иногда связывание осуществляется неявно (внутри функций connect и accept), но выполнять его необходимо во всех случаях.
- Вид адреса зависит от выбранного вами домена.
  - В Unix-домене это текстовая строка - имя файла, через который происходит обмен данными.
  - В Internet-домене адрес задаётся комбинацией IP-адреса и 16-битного номера порта. IP-адрес определяет хост в сети, а порт - конкретный сокет на этом хосте.

Функция привязки сокета с выбранным адресом

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

- **sockfd** – дескриптор сокета
- **addr** – указатель на структуру адреса
- **addlen** – длина структуры

Структура адерса sockaddr_in

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

Поле **sa_family** содержит идентификатор домена, тот же, что и первый параметр функции **socket**. В зависимости от значения этого поля по-разному интерпретируется содержимое массива **sa_data**.

Структура адерса sockaddr_in

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

Структура in_addr

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

- Компьютер может иметь несколько сетевых интерфейсов.
- Если мы готовы принимать соединения из любого интерфейса, то следует использовать константу **INADDR_ANY**.

Порядок следования байтов

- Существует два порядка хранения байтов в слове и двойном слове. Один из них называется *порядком хоста* (host byte order), другой - *сетевым порядком* (network byte order) хранения байтов.
- При указании IP-адреса и номера порта необходимо преобразовать число из порядка хоста в сетевой. Для этого используются функции **htons** (Host TO Network Short) и **htonl** (Host TO Network Long). Обратное преобразование выполняют функции **ntohs** и **ntohl**.

Слушание сокета

```
int listen(int sockfd, int backlog);
```

- Создаёт очередь запросов на соединение, сокет переводится в режим ожидания соединения клиентов.
- **sockfd** – дескриптор сокета
- **backlog** – длина очереди запросов

Системный вызов select()

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int n, fd set *readfds, fd set *writefds,
fd_set *exceptfds, struct timeval *timeout);
```

Функция **select** работает с тремя множествами дескрипторов, каждое из которых имеет тип **fd_set**.

- В множество **readfds** записываются дескрипторы сокетов, из которых требуется читать данные (слушающие сокеты добавляются в это же множество).
- Множество **writefds** содержит дескрипторы сокетов, в которые мы собираемся писать;
- **exceptfds** - дескрипторы сокетов, которые нужно контролировать на возникновение ошибки.
- Если хотя бы один сокет готов к выполнению заданной операции, **select** возвращает ненулевое значение, а все дескрипторы, которые привели к "срабатыванию" функции, записываются в соответствующие множества.

Множества дескрипторов

- **FD_ZERO(fd_set *set)** - очищает множество **set**
- **FD_SET(int fd, fd_set *set)** - добавляет дескриптор **fd** в множество **set**
- **FD_CLR(int fd, fd_set *set)** - удаляет дескриптор **fd** из множества **set**
- **FD_ISSET(int fd, fd_set *set)** - проверяет, содержится ли дескриптор **fd** в множестве **set**

Системный вызов accept()

```
#include <sys/socket.h>
int accept(int sockfd, void *addr, int
*addrlen);
```

Если сервер готов обслужить очередной запрос, используется функция accept();

sock.

- Функция возвращает **дескриптор** *нового* сокета.
- **sockfd** – слушающий сокет
- **addr** – структура адреса сокета клиента
- **addrlen** – длина этой структуры

Отправка (send) и получение (recv) данных

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf, int len, int flags);
```

- **sockfd** – дескриптор сокета
- **msg** – указатель на буфер с данныим
- **len** – длина данных
- **flags** – дополнительные флаги (можно оставить NULL)

Закрытие сокета

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

После окончания работы с сокетом, его необходимо закрыть.

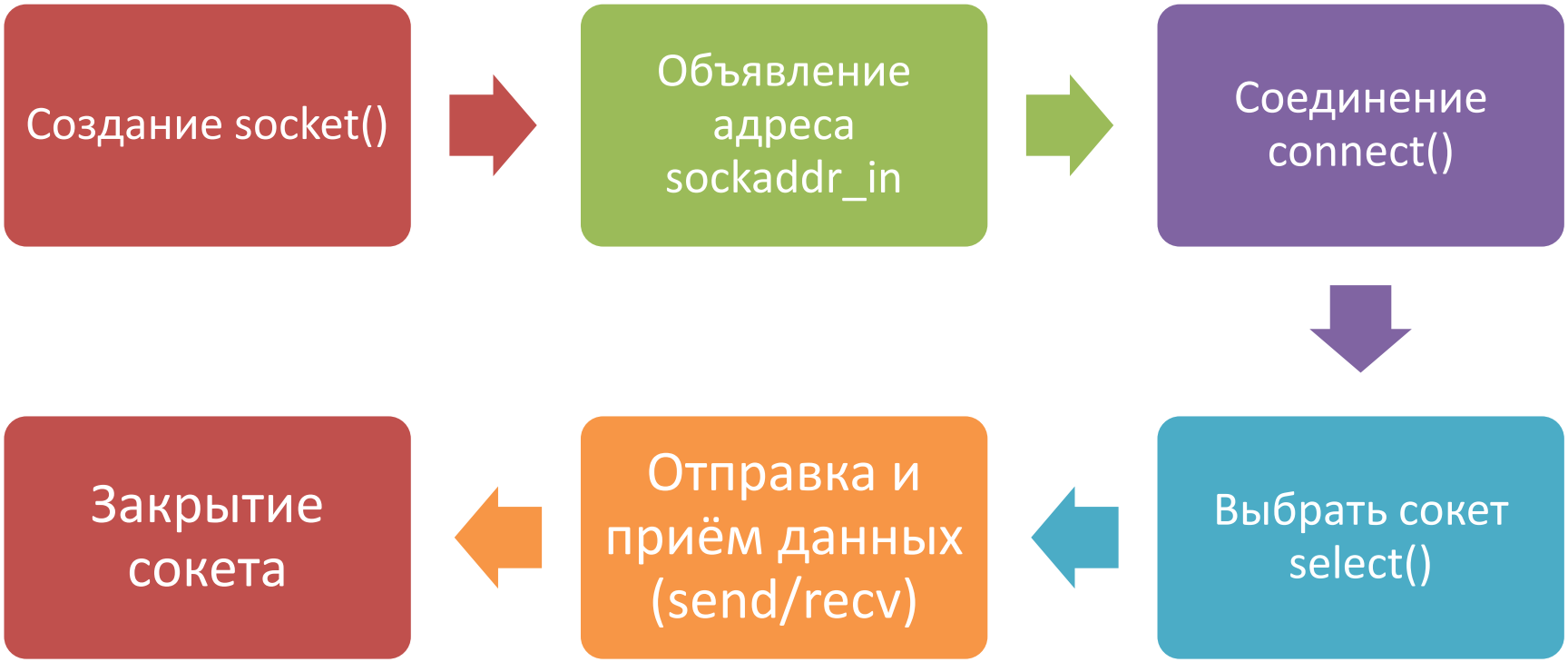
Установка соединения (сервер)

Установка соединения connect()

```
#include <sys/types.h>
#include <sys/socket.h>

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

Установка соединения (клиент)



Функция fflush()

```c
#include <stdio.h>  // Для printf, fflush
#include <unistd.h>  // Для sleep

int main (void)
{
   int i=0; // Счетчик секунд
   // Бесконечный цикл
   while (1)
   {
      // Вывод строки (строка записывается в буфер)
      printf ("\r%d",i);
      //Сброс буфера (строка отобразится в консоле)
      fflush (stdout);
      //Задержка на 1 секунду
      sleep (1);
      //Увеличение счетчика секунд на 1
      i++;
   }
   return 0;
}
```

Функции setsockopt() и getsockopt()

```
int yes = 1;
setsockopt(*sockfd, SOL_SOCKET,
     SO_REUSEADDR, &yes, sizeof(int));
```

Пример сервера server.c

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

#define PORT 4950
#define BUFSIZE 1024

void send to all(int j, int i, int sockfd, int nbytes_recvd, char
*recv_buf, fd_set *master)
{
        if (FD_ISSET(j, master)){
                if (j != sockfd && j != i) {
                        if (send(j, recv_buf, nbytes_recvd, 0) == -1)
{
                                perror("send");
                        }
                }
        }
}
```

Пример сервера

server.c

```
void send_recv(int i, fd_set *master, int sockfd, int fdmax)
{
        int nbytes_recvd, j;
        char recv_buf[BUFSIZE], buf[BUFSIZE];

        if ((nbytes_recvd = recv(i, recv_buf, BUFSIZE, 0)) <= 0) {
                if (nbytes_recvd == 0) {
                        printf("socket %d hung up\n", i);
                }else {
                        perror("recv");
                }
                close(i);
                FD_CLR(i, master);
        }else {
        //      printf("%s\n", recv_buf);
                for(j = 0; j <= fdmax; j++){
                        send to all(j, i, sockfd, nbytes_recvd,
                                recv_buf, master );
                }
        }
}
```

Пример сервера server.c

```
void connection_accept(fd_set *master, int *fdmax, int sockfd,
struct sockaddr_in *client_addr)
{
        socklen_t addrlen;
        int newsockfd;

        addrlen = sizeof(struct sockaddr_in);
        if((newsockfd = accept(sockfd, (struct sockaddr
*)client_addr,                          &addrlen)) == -1) {
                perror("accept");
                exit(1);
        }else {
                FD_SET(newsockfd, master);
                if(newsockfd > *fdmax){
                        *fdmax = newsockfd;
                }
                printf("new connection from %s on port %d \n",
                        inet_ntoa(client_addr->sin_addr),
                                ntohs(client_addr->sin_port));
        }
}
```

Пример сервера server.c

```
void connect_request(int *sockfd, struct sockaddr_in *my_addr)
{
        int yes = 1;

        if ((*sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }

        my_addr->sin_family = AF_INET;
        my_addr->sin_port = htons(4950);
        my_addr->sin_addr.s_addr = INADDR_ANY;
        memset(my_addr->sin_zero, '\0', sizeof my_addr->sin_zero);

        if (setsockopt(*sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
sizeof(int)) == -1) {
                perror("setsockopt");
                exit(1);
        }
```

Пример сервера
server.c

```
        if (bind(*sockfd, (struct sockaddr *)my_addr,
sizeof(struct sockaddr)) == -1) {
                perror("Unable to bind");
                exit(1);
        }
        if (listen(*sockfd, 10) == -1) {
                perror("listen");
                exit(1);
        }
        printf("\nTCPServer Waiting for client on port 4950\n");
        fflush(stdout);
}
```

Пример сервера server.c

```
int main()
{
        fd_set master;
        fd_set read_fds;
        int fdmax, i;
        int sockfd = 0;
        struct sockaddr_in my_addr, client_addr;

        FD_ZERO(&master);
        FD_ZERO(&read_fds);
        connect_request(&sockfd, &my_addr);
        FD_SET(sockfd, &master);

        fdmax = sockfd;
        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }
```

```
        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }

                for (i = 0; i <= fdmax; i++){
                        if (FD_ISSET(i, &read_fds)){
                                if (i == sockfd)
                                        connection_accept(&master,
                                &fdmax, sockfd, &client_addr);
                                else
                                        send_recv(i, &master, sockfd,
                                        fdmax);
                        }
                }
        }
        return 0;
}
```

Пример сервера server.c

Пример клиента client.c

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

#define BUFSIZE 1024

void send_recv(int i, int sockfd)
{
        char send_buf[BUFSIZE];
        char recv_buf[BUFSIZE];
        int nbyte_recvd;

        if (i == 0){
                fgets(send_buf, BUFSIZE, stdin);
                if (strcmp(send_buf , "quit\n") == 0) {
```

Пример клиента client.c

```
void send_recv(int i, int sockfd)
{
        char send_buf[BUFSIZE];
        char recv_buf[BUFSIZE];
        int nbyte_recvd;

        if (i == 0){
                fgets(send_buf, BUFSIZE, stdin);
                if (strcmp(send_buf , "quit\n") == 0) {
                        exit(0);
                }else
                        send(sockfd, send_buf, strlen(send_buf), 0);
        }else {
                nbyte_recvd = recv(sockfd, recv_buf, BUFSIZE, 0);
                recv_buf[nbyte_recvd] = '\0';
                printf("%s\n" , recv_buf);
                fflush(stdout);
        }
}
```

Пример клиента client.c

```
void connect_request(int *sockfd, struct sockaddr_in
*server_addr)
{
        if ((*sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }
        server_addr->sin_family = AF_INET;
        server_addr->sin_port = htons(4950);
        server_addr->sin_addr.s_addr = inet_addr("217.9.88.22");
        memset(server_addr->sin_zero, '\0', sizeof server_addr-
>sin_zero);

        if(connect(*sockfd, (struct sockaddr *)server_addr,
sizeof(struct sockaddr)) == -1) {
                perror("connect");
                exit(1);
        }
}
```

Пример клиента client.c

```
int main()
{
        int sockfd, fdmax, i;
        struct sockaddr_in server_addr;
        fd_set master;
        fd_set read_fds;

        connect_request(&sockfd, &server_addr);
        FD_ZERO(&master);
        FD_ZERO(&read_fds);
        FD_SET(0, &master);
        FD_SET(sockfd, &master);
        fdmax = sockfd;

        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }
```

Пример клиента client.c

```
        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }

                for(i=0; i <= fdmax; i++ )
                        if(FD_ISSET(i, &read_fds))
                                send_recv(i, sockfd);
        }
        printf("client-quited\n");
        close(sockfd);
        return 0;
}
```

Литература

1. Программирование сокетов в Linux - https://rsdn.org/article/unix/sockets.xml
2. Multi client chat server in c - https://vidyakv.wordpress.com/2011/11/29/multi-client-chat-server-in-c/
3. Таненбаум Э. С., Херберт Б. Современные операционные системы. 4-е изд. – «Издательский дом “Питер”», 2015
4. Кейно П. П. Разработка архитектуры программного комплекса синхронизатора при интерпретаторе декларативного языка BML // Прикладная информатика. — 2016. — 2(52). — С. 65-77.