Система ввода/вывода в Linux

2021

Блокирующий и неблокирующий ввод/вывод

Сокеты

Каналы

Блокирующий ввод/вывод

Версия 1:

```
int fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC, 0644);
const char msg[] = "hello\n";
ssize_t nbytes = write(fd, msg, sizeof(msg));
close(fd);
```

Версия 2:

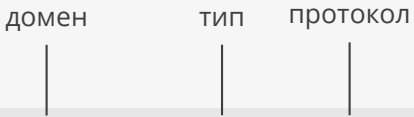
```
...
ssize_t nwritten = 0;
while (nwritten != sizeof(msg));
    ssize_t nbytes = write(fd, msg+nwritten, sizeof(msg)-nwritten);
    if (nbytes == -1) { throw ... } // ошибка ввода/вывода
    nwritten += nbytes;
}
...
```

Неблокирующий ввод/вывод

Версия 3:

```
int fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC|O_NONBLOCK, 0644);
...
ssize_t nwritten = 0;
while (nwritten != sizeof(msg));
    ssize_t nbytes = write(fd, msg+nwritten, sizeof(msg)-nwritten);
    if (nbytes == -1 && errno != EAGAIN) { throw ... }
    if (nbytes > 0) { nwritten += nbytes; }
}
...
```

Блокирующий сокет



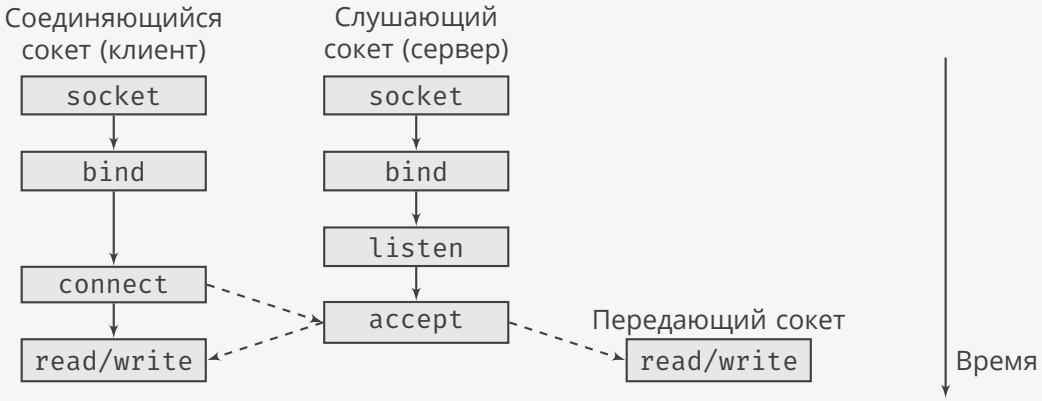
Серверная часть:

```
int fd = socket(AF_INET, SOCK_STREAM, 0); // создать сокет
sockaddr_in addr{};           // инициализация нулями
addr.sin_family = AF_INET;    // IPv4
addr.sin_port = htons(22222);
addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK); // 127.0.0.1
// или inet_aton("127.0.0.1", &addr.sin_addr.s_addr);
bind(fd, &addr, sizeof(addr)); // присвоить адрес (имя) сокету
listen(fd, SOMAXCONN);         // ожидать соединений
int client_fd = accept(fd, nullptr, nullptr); // принять соединение
...
shutdown(fd, SHUT_RDWR);       // выключить чтение/запись
close(fd); close(client_fd);   // закрыть файловый дескрипторы
// hton* — сетевой порядок байт для uint*_t
```

Блокирующий сокет

Клиентская часть:

```
int fd = socket(AF_INET, SOCK_STREAM, 0); // создать сокет
sockaddr_in addr{};                       // инициализация нулями
addr.sin_family = AF_INET;                // IPv4
addr.sin_port = htons(22222);
addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK); // 127.0.0.1
sockaddr_in any{};                // инициализация нулями
bind(fd, &any, sizeof(any));      // присвоить адрес (имя) сокету
// или inet_aton("127.0.0.1", &addr.sin_addr.s_addr);
connect(fd, &addr, sizeof(addr));
...
shutdown(fd, SHUT_RDWR);          // выключить чтение/запись
close(fd);                        // закрыть файловый дескриптор
// inet_aton — строка в IP-адрес
```


Соединяющийся сокет (клиент)
Слушающий сокет (сервер)
socket
bind
connect
read/write
socket
bind
listen
accept
Передающий сокет
read/write
Время

Неблокирующий сокет

Серверная часть:

```
int server_fd = socket(AF_INET, SOCK_STREAM|SOCK_NONBLOCK, 0);
bind(...); listen(...);
std::vector<pollfd> fds{{server_fd,POLLIN}}; // добавить сервер
poll(fds.data(), fds.size(), -1); // опросить дескрипторы
for (size_t i=0; i<fds.size(); ++i) {
  const pollfd& e = fds[i];
  if (f.revents == 0) { continue; }
  if (f.fd == server_fd) { // прием новых соединений
    int client_fd = accept4(server_fd, 0, 0, SOCK_NONBLOCK);
    fds.push_back({client_fd, POLLIN|POLLOUT}); // добавить клиента
  } else { ... } // чтение/запись в существующие соединения
}
```

Структура из `poll.h`:

```
struct pollfd { int fd; short events; short revents; };
```

Неблокирующий сокет

Клиентская часть:

```
int fd = socket(AF_INET, SOCK_STREAM|SOCK_NONBLOCK, 0);
bind(...); connect(...);
std::vector<pollfd> fds{{fd,POLLIN}}; // добавить дескриптор
poll(fds, fds.size(), -1);            // опросить дескрипторы
for (const pollfd& f : fds) {
  if (f.revents == 0) { continue; }
  if (f.revents & POLLIN) { ... }     // чтение
  if (f.revents & POLLOUT) { ... }    // запись
  if (f.revents & POLLHUP) { ... }    // бросили трубку
}
```

Проблемы `poll`:

- сложно использовать из нескольких потоков,
- медленно работает при большом количестве дескрипторов.

Альтернативы:

- epoll (Linux),
- `kqueue` (FreeBSD, MacOS).

Неблокирующий сокет с epoll

Структура данных для epoll:

```
typedef union epoll_data {
    void        *ptr;
    int          fd;
    uint32_t     u32;
    uint64_t     u64;
} epoll_data_t;                 // аналог std::any

struct epoll_event {
    uint32_t     events;        // события epoll
    epoll_data_t data;          // пользовательские данные
};
```

Неблокирующий сокет с epoll

Серверная часть:

```
int server_fd = socket(...); bind(...); listen(...);
int epfd = epoll_create1(0);
epoll_event event{EPOLLIN,{server_fd}}; // добавить сервер
epoll_ctl(epfd, EPOLL_CTL_ADD, server_fd, &event);
std::vector<epoll_event> events(100); // контейнер для событий
int nevents = epoll_wait(epfd, events.data(), events.size(), -1);
for (int i=0; i<nevents; ++i) {
    const epoll_event& e = events[i];
    if (e.data.fd == server_fd) { // прием новых соединений
        int client_fd = accept4(...);
        epoll_event event{EPOLLIN, {client_fd}}; // новый клиент
        epoll_ctl(epfd, EPOLL_CTL_ADD, client_fd, &event);
    } else { ... } // чтение/запись в существующие соединения
}
close(epfd); // закрыть дескриптор epoll
```

Неблокирующий сокет с epoll

Клиентская часть:

```
int fd = socket(...); bind(...); connect(...);
epoll_event event{EPOLLIN,{fd}};
epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &event); // добавить дескриптор
std::vector<epoll_event> events(100); // контейнер для событий
int nevents = epoll_wait(epfd, events.data(), events.size(), -1);
for (int i=0; i<nevents; ++i) {
    const epoll_event& e = events[i];
    if (e.events & EPOLLIN) { ... }   // чтение
    if (e.events & EPOLLOUT) { ... }  // запись
    if (e.events & EPOLLHUP) { ... }  // бросили трубку
}
close(epfd); // закрыть дескриптор epoll
```

Три способа мультиплексирования ввода/вывода:

	`poll`	`epoll`	`select`
структура данных	массив	хэш-таблица	устарел
где хранится	процесс	ядро	
нужно дескрипторов	≤ 1000	> 1000	
несколько потоков	сложно	ок	

Параметры сокетов

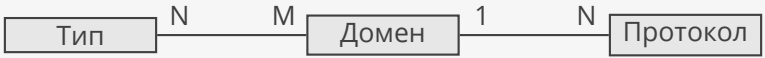
```
// повторное использование IP-адреса
int value = 1;
setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &value, sizeof(value));
                уровень       опция

// локальный адрес (из вызова bind)
union myaddress { sockaddr_in sain; sockaddr sa; };
myaddress addr; socklen_t len;
getsockname(fd, &addr.sa, &len);

// удаленный адрес (из вызова accept)
getpeername(fd, &addr.sa, &len);

// неблокирующий ввод/вывод (переносимый код)
int old_flags = fcntl(fd, F_GETFL);       // получить флаги
fcntl(fd, F_SETFL, old_flags|O_NONBLOCK); // добавить флаг
```

Домены, типы и протоколы



Домен	`AF_UNIX`	локальные соединения
	`AF_INET`	TCP/IPv4
	`AF_INET6`	TCP/IPv6
	`AF_NETLINK`	взаимодействие с ядром
Тип	`SOCK_STREAM`	с установкой соединения
	`SOCK_DGRAM`	без установки соединения
	`SOCK_RAW`	прямой доступ к пакетам
Протокол	`0`	протокол по умолчанию

Каналы

Создать канал:

```
int pipe_fd[2];                    // fd[0] — чтение, fd[1] — запись
pipe2(pipe_fd, O_NONBLOCK);        // создание канала
// ...
close(pipe_fd[0]);
close(pipe_fd[1]);
```

Направить стандартный вывод в канал:

```
dup2(pipe_fd[1], STDOUT_FILENO); // перенаправление в канал
```

Перемещение страниц памяти

Отдать страницы памяти каналу:

```
char buf[4096*10];
iovec v{buf, sizeof(buf)};
vmsplice(pipe_fd[1], &v, 1, SPLICE_F_GIFT|SPLICE_F_NONBLOCK);
```

Переместить страницы из канала в сокет:

```
splice(pipe_fd[0], 0, sock_fd, 0, sizeof(buf), SPLICE_F_MOVE);
```

Переместить страницы из канала в файл:

```
int file_fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC, 0644);
ftruncate(file_fd, 4096*10);
void* ptr = mmap(0, 4096*10, PROT_WRITE, MAP_SHARED, file_fd, 0);
splice(sock_fd, 0, pipe_fd[1], 0, 4096*10, SPLICE_F_MOVE);
splice(pipe_fd[0], 0, file_fd, 0, 4096*10, SPLICE_F_MOVE);
```

Перемещение страниц памяти

Переместить страницы из файла в сокет:

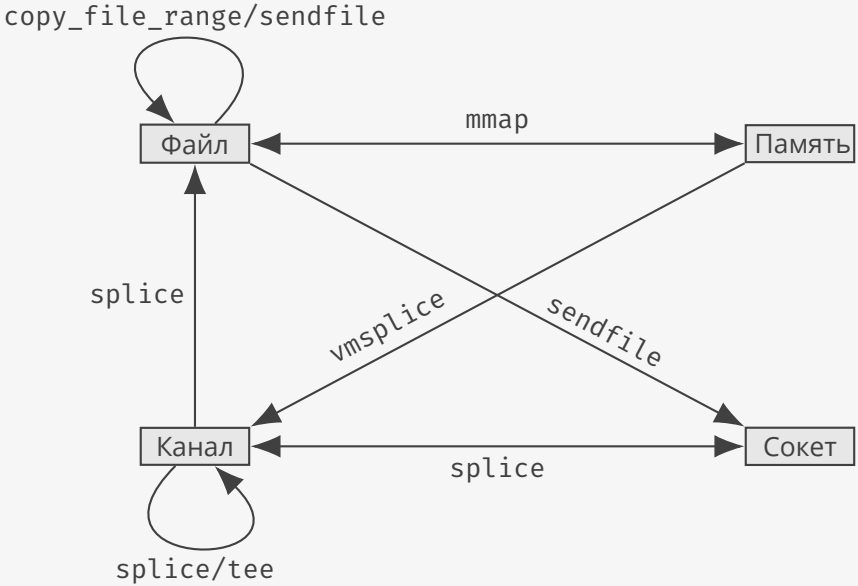
```
sendfile(sock_fd, file_fd, 0, 4096*10);
```

Переместить страницы из файла в файл:

```
copy_file_range(file_fd_1, 0, file_fd_2, 0, 4096*10, 0);
```

«Скопировать» страницы из канала в канал:

```
tee(pipe_fd_1[0], pipe_fd_2[1], 4096*10, 0);
```


copy_file_range/sendfile
Файл
mmap
Память
splice
vmsplice
sendfile
Канал
Сокет
splice
splice/tee

Преимущества `splice/sendfile/vmsplice/tee` перед `read/write`:

- Ядро перемещает страницы без копирования, если это возможно.
- Копирование данных всегда происходит внутри ядра.

Ссылки

- Пример poll.
- Пример epoll.
- Примеры splice/vmsplice.