

# Взаимодействие процессов в Linux

2021



# Сигналы

Отправка сигнала:

```
pid_t pid = 3333; // номер процесса
kill(pid, SIGTERM); // завершить процесс
kill(pid, SIGINT); // прерывание с клавиатуры
kill(pid, SIGKILL); // убить процесс
kill(pid, SIGSEGV); // ошибка по адресу
kill(pid, SIGCHLD); // сигнал от дочернего процесса
kill(pid, 0); // жив ли процесс?
```

Прием сигнала:

```
void my_handler(int sig) { ... } // обработка сигнала

sigaction a{}; // заполнение нулями
s.sa_handler = my_handler;
sigaction(SIGTERM, &s, nullptr);
```

# Ограничения обработчиков сигналов

Глобальные переменные:

```
void my_handler() { // обработчик сигнала
    std::cout << "Hello from handler\n"; // катастрофа
}
std::cout << "Hello from main\n"; // основная программа
```

Системные вызовы:

```
int ret;
do {
    ret = epoll_wait(...);
} while (ret == -1 && errno == EINTR); // ручной перезапуск
```

Много потоков:

```
kill(pid, SIGTERM); // отправить любому потоку в процессе
tkill(tid, SIGTERM); // отправить конкретному потоку
```

# Каналы

Чтение вывода другой программы:

```
int pipe_fd[2];
pipe(pipe_fd); // создание канала
pid_t pid = fork(); // дескрипторы наследуются
if (pid == 0) {
    close(pipe_fd[0]); // закрыть на чтение
    dup2(pipe_fd[1], STDOUT_FILENO); // перенаправление
    char* const argv[] = {"date", 0}; // стандартного вывода
    execvp(argv[0], argv);
    exit(0);
}
close(pipe_fd[1]); // закрыть на запись
... // считать вывод date из pipe_fd[0]
```

Сигнал SIGPIPE:

```
int pipe_fd[2];  
pipe(pipe_fd);  
close(pipe_fd[0]); // закрыть на чтение  
const char buf[] = "hello";  
write(pipe_fd[1], buf, sizeof(buf)-1);  
... // программа завершается по сигналу SIGPIPE
```

Игнорирование сигнала SIGPIPE:

```
sigaction a{};  
s.sa_handler = SIG_IGN;  
sigaction(SIGPIPE, &s, nullptr);
```

# Именованные каналы

Создание именованного канала (чтение):

```
mkfifo("/tmp/myfifo", 0666);  
int fd = open("/tmp/myfifo", O_RDONLY);  
... // чтение  
close(fd);
```

Открытие именованного канала (запись):

```
int fd = open("/tmp/myfifo", O_WRONLY);  
... // запись  
close(fd);
```

# Локальные сокеты

Именованный сокет (сервер):

```
int fd = socket(AF_UNIX, SOCK_STREAM, 0);
union { sockaddr sa; sockaddr_un saun{}} a;
char name[] = "/tmp/mysocket"; // имя сокета в файловой системе
a.saun.sun_family = AF_UNIX;
std::copy_n(name, sizeof(a.saun.sun_path)-1, a.saun.sun_path);
bind(fd, &a.sa, sizeof(sockaddr_un));
listen(fd, SOMAXCONN);
... // обслуживание клиентов
close(fd);
```

Скрытый сокет (сервер):

```
...
char name[] = "\0/tmp/mysocket"; // начинается на нулевой байт
...
```

# Получение пользователя и группы

Клиент:

```
int fd = socket(...);
int one = 1;
setsockopt(fd, SOL_SOCKET, SO_PASSCRED, &one, sizeof(one));
...
```

Сервер:

```
int fd = socket(...);
bind(fd, ...); listen(fd, ...);
int client_fd = accept(...);
ucred uc; // процесс, пользователь, группа
socklen_t size = sizeof(uc);
getsockopt(client_fd, SOL_SOCKET, SO_PEERCRED, &uc, &size);
std::cout << "pid = " << uc.pid << '\n';
std::cout << "uid = " << uc.uid << '\n';
std::cout << "gid = " << uc.gid << '\n';
```



# Низкоуровневые сообщения

Произвольное сообщение:

```
int fd = socket(...); bind(...); connect(...);  
char mydata[] = "hello"; // данные для отправки  
iovec v{mydata, sizeof(mydata)};  
msg_hdr msg; // контрольное сообщение + полезные данные  
msg.msg_control = nullptr; // начало контрольного сообщения  
msg.msg_controllen = 0; // размер контрольного сообщения  
msg.msg_iov = &v; // полезные данные  
msg.msg_iovlen = 1;  
sendmsg(fd, &m, 0);
```

# Отправка файловых дескрипторов

```
...  
const size_t n = 3*sizeof(int);  
union { cmsghdr m; char data[MSG_SPACE(n)] } cmsg;  
cmsg.m.cmsg_len = MSG_LEN(n);  
cmsg.m.cmsg_level = SOL_SOCKET;  
cmsg.m.cmsg_type = SCM_RIGHTS; // отправка дескрипторов  
msg.msg_control = cmsg.data; // привязка контрольного сообщения  
msg.msg_controllen = MSG_SPACE(n);  
int* data = MSG_DATA(&cmsg.m);  
data[0] = 0; data[1] = 1; data[2] = 2; // дескрипторы  
...
```

# Отладка сокетов

```
$ ss -x # показать локальные сокет (AF_UNIX)
... @/tmp/.X11-unix/X0 ... # абстрактный локальный сокет (X11)

$ ss -t # показать TCP сокет (AF_INET*)
... 172.27.111.111:54602 ... # сетевое соединение (SSH)

$ ss -tl # показать слушающие сокет
... 0.0.0.0:22 ... # TCP сокет (SSH)
```

# Виртуальная общая память

Именованная область памяти:

```
int fd = shm_open("/mymem", O_CREAT|O_RDWR, 0644);  
void* ptr = mmap(..., fd, 0);  
...  
munmap(ptr, ...);  
close(fd);  
shm_unlink("/mymem");
```

Имя в файловой системе:

```
$ ls /dev/shm  
mymem
```

Нет копирования в буфер ядра, но нужна синхронизация.

# Общая память как буфер

Идея: минимизировать взаимодействие с ядром.

```
struct SharedMemoryHeader {
    Semaphore read_semaphore, write_semaphore;
    size_t read_offset = 0, write_offset = 0;
};
class SharedMemoryBuffer {
    void* ptr; // указатель на общую память
    SharedMemoryHeader* header; // заголовок
    bool owner; // является ли процесс владельцем
public:
    SharedMemoryBuffer(bool owner): ptr(...), owner(owner) {
        header = owner ? new (ptr) SharedMemoryHeader
            : static_cast<SharedMemoryHeader*>(ptr);
        header->write_offset = sizeof(SharedMemoryHeader);
        header->read_offset = sizeof(SharedMemoryHeader);
    }
};
```

Синхронизация доступа к буферу:

```
void SharedMemoryBuffer::lock() {  
    if (owner) { header->read_semaphore.wait(); }  
    else { header->write_semaphore.wait(); }  
}  
void SharedMemoryBuffer::unlock() {  
    if (owner) { header->write_semaphore.notify_one(); }  
    else { header->read_semaphore.notify_one(); }  
}
```

# Устаревшие системные вызовы

```
shmget // виртуальная общая память  
shmctl  
semget // семафоры  
semctl  
msgget // очереди сообщений  
msgsnd  
msgrcv
```

# Сравнительная таблица

Тип	Передача данных	Способ	Уровень сложности
Сигналы	нет	1:n	низкий
Каналы	да	1:1	низкий
Сокеты	да	m:n	средний
Общая память	да	m:n	высокий



# Ссылки

- ▶ [Сигналы.](#)
- ▶ [Каналы.](#)
- ▶ [Именованные каналы.](#)
- ▶ [Локальные сокеты.](#)
- ▶ [Общая память.](#)
- ▶ [Семафоры.](#)
- ▶ [Устаревшие системные вызовы.](#)