

ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

05.04.19

Стандартная C библиотека (libc.so)


- Взаимодействие с ОС прямо или косвенно осуществляется через libc.
- Что входит в libc: библиотека языка C (malloc, free, strcmp), библиотека для Unix (стандарт POSIX) (open, write, read), поддержка многопоточности (pthread), математические функции и др.
- Заголовочные файлы stdio.h, stdlib.h, string.h и многие другие являются частью стандартной библиотеки.
- libc.so выполняет первичную инициализацию процесса, а также завершение процесса. Первая исполняемая функция - `_start`. `_start` в итоге вызывает `main`.
- Существуют разные реализации: glibc (на ПК), musl (на встроенных системах) и др.

Системные вызовы

`printf` = формирование строки в буфере и вывод этой строки через функцию `write`. Как реализована `write`?

- Функции и код ядра Linux защищены аппаратно и не доступны приложениям.
- Системные вызовы - программные прерывания, т.е. имеют аппаратную поддержку в процессоре в виде специальных инструкций.
- В целях безопасности ядро строго контролирует передаваемые аргументы.
- Системные вызовы реализуются с помощью кода на ассемблере. Приложения вызывают тонкие обертки из `libc`.
- Основное взаимодействие между приложением и ядром осуществляется через системные вызовы (`open`, `close`, `read`, `write`, `brk` и др.).


API для работы с файлами в ОС Linux



```
int open(const char *pathname, int flags);
```

Функция `open` возвращает файловый дескриптор. Флаги задают режим доступа:

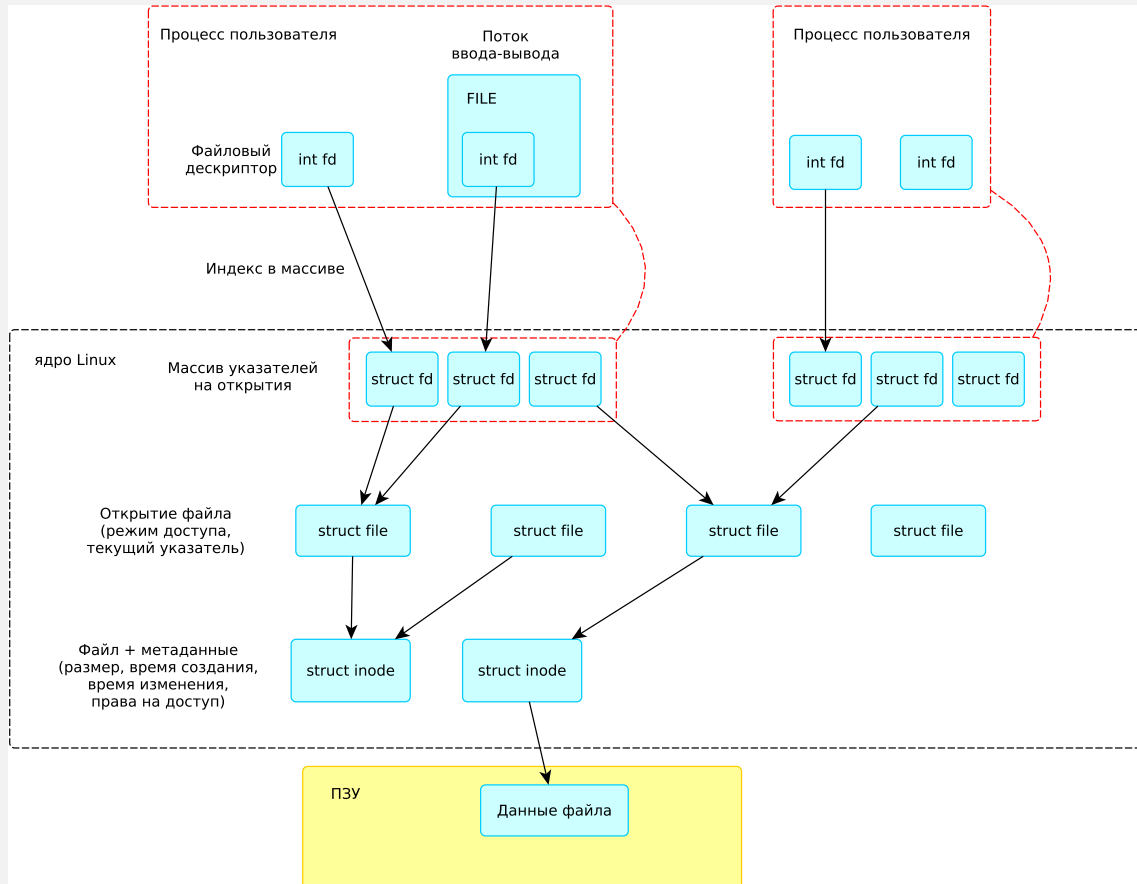
- `O_RDONLY` - только чтение,
- `O_WRONLY` - только запись,
- `O_RDWR` - чтение и запись,
- `O_TRUNC` - удаление всех данных из файла.



```
ssize_t read(int fd, void *buf, size_t nbyte);  
ssize_t write(int fd, const void *buf, size_t nbyte);
```

Чтение в буфер `buf` или запись из буфера `buf`. Количество - `nbyte` байтов.

Что такое файл и файловый дескриптор



Замечания

- У каждого процесса свое пространство файловых дескрипторов, совпадение номеров не означает совпадение файлов.
- API работает с дескриптором, а не с файлом, т.е. файл может быть виртуальным. Пример полиморфизма.
- Зачем нужны указатели (с дескрипторами) на открытия? Чтобы иметь несколько указателей (в том числе из разных процессов) на одно и то же открытие.
- Дескриптор 0 (соответствует потоку stdin) - стандартный поток ввода, 1 (stdout) - поток вывода, 2 (stderr) - поток сообщений об ошибках.
- printf записывает сообщение в дескриптор 1. printf(...) = fprintf(stdout, ...).
- Имена и пути к файлам - отдельный вопрос. Файл может существовать даже без пути к нему.

Потоки ввода-вывода

■

```
FILE *fopen(const char *path, const char *mode);
```

Открытие файла. `mode` - режим доступа, "r" - чтение, "w" - запись с очисткой, "r+" и "w+" - чтение и запись.

■

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Чтение в буфер `ptr` или запись из буфера `ptr`. Объем данных в байтах = размер блока `size`, умноженный на количество блоков `nmemb`.

■ Чтение и запись буферизируются.

■ Поток ввода-вывода содержит файловый дескриптор. На низком уровне чтение и запись производятся через `read` и `write`.

Создание нового процесса


Запуск приложения = создание нового процесса + исполнение приложения в этом процессе

```
pid_t fork(void);
```

Что делает `fork`.


- Создает новый процесс, который исполняет то же приложение с момента вызова `fork`.
- Для нового процесса выделяется идентификатор (PID, см. вывод команды `ps`).
- Новый процесс является дочерним, тот, который вызвал `fork` - родительским (см. вывод команды `pstree`).
- Дочерний процесс наследует от родительского открытые файлы и идентификатор пользователя.

Исполнение приложения




```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

Функция заменяет текущее приложение на новое и передает аргументы командной строки.



```
void exit(int status);
```

Завершение процесса с заданным кодом. Посмотреть код из shell: `echo $?`



Функция `waitpid` позволяет дождаться завершения дочернего процесса с заданным PID.



```
int system(const char *command);
```

Функция создает новый процесс, запускает в нем `/bin/sh` с командой `command` и дожидается его завершения.

Аргументы командной строки

```
$ ls --width 50 -l home/student/
```

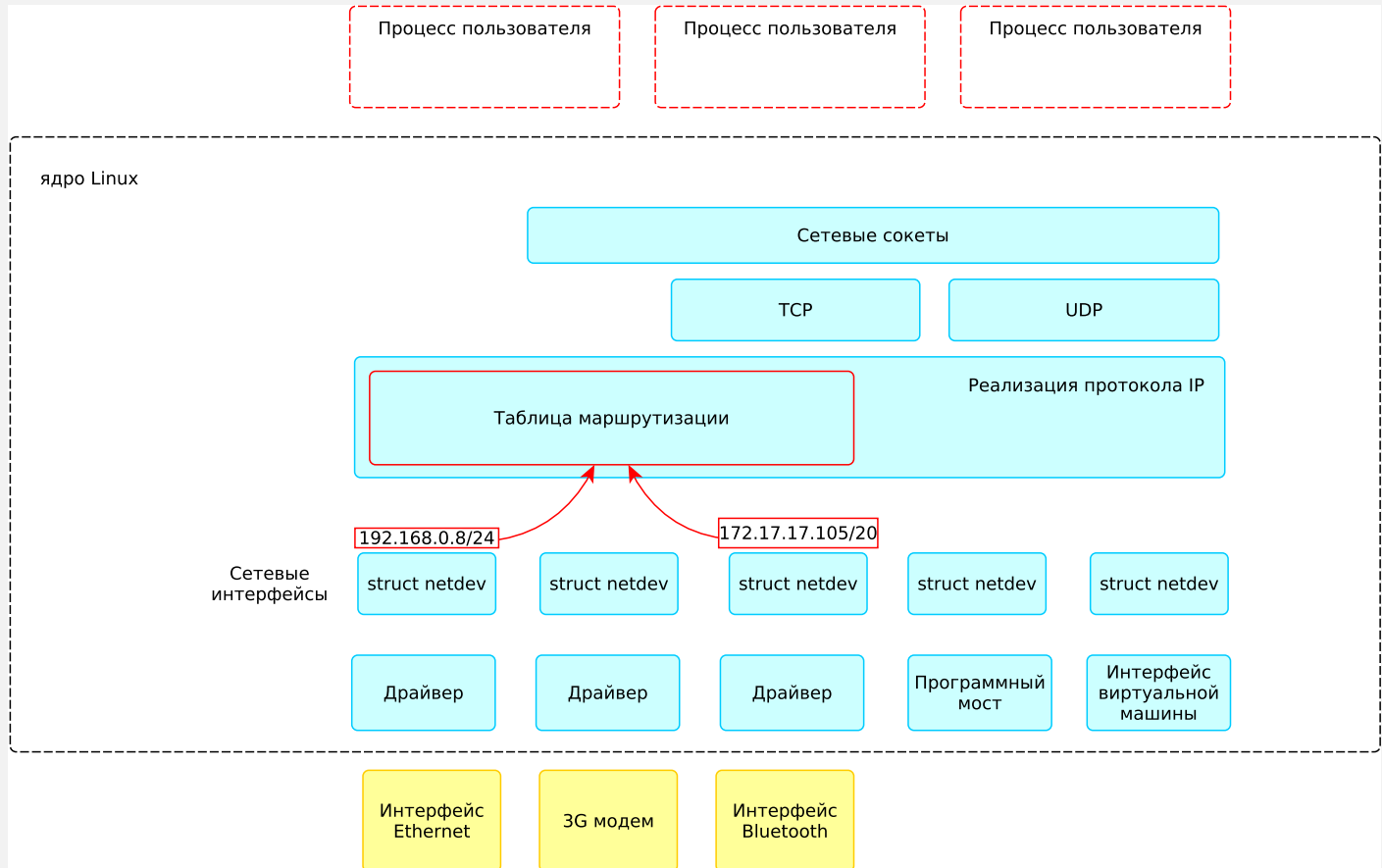
- Позиционные аргументы (синий) - порядок следования определяет смысл.
- Именованные аргументы (красные) - имя аргумента задается явно в короткой (-w) или длинной (--width) форме.
- Обычно именованные аргументы являются необязательными.
- У именованного аргумента может быть значение.
- С точки зрения ОС все аргументы - это массив строк. Классификацию аргументов осуществляет приложение.

Анализ именованных аргументов с помощью getopt

```
int opt;
while ((opt = getopt(argc, argv, "cLd:")) != -1) {
    switch (opt) {
        case 'c':
            has_c = true; break;
        case 'd':
            debug_level = atoi(optarg); break;
        ...
    }
}
```

- Поддерживается только короткая форма именованных аргументов.
- getopt вызывается в цикле. На каждой итерации анализируется один аргумент.
- Поддерживаются аргументы со значением (: в дескрипторе). Строка со значением доступна через указатель optarg.
- Меняет порядок аргументов (только в glibc!!!). Все позиционные аргументы переносятся в конец argv.

Сетевой стек Linux



Сетевые интерфейсы

```
$ ip addr
```

```
...
```

```
enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master lxcbr1 state UP  
    link/ether 08:60:6e:81:12:ac brd ff:ff:ff:ff:ff:ff  
    inet 172.17.17.105/24 brd 172.17.17.255 scope global enp3s0
```

- Флаг NO-CARRIER - отсутствие физического („несущего”) сигнала.
- Флаг UP -- административное включение интерфейса.
- Состояние UP/DOWN - наличие/отсутствие соединения.
- MTU - maximum transmission unit, максимальный размер пакета, который можно передать без фрагментации.
- IP-адрес (172.17.17.105/24) - добавляется в таблицу маршрутизации (вывод по команде `ip route`). Маска (/24) определяет множество адресов, которые напрямую достижимы через данный интерфейс.
- MAC-адрес - адрес интерфейса в локальной сети Ethernet.

Настройка интерфейсов через iproute2

- Удаление всех IP-адресов с интерфейса `ge1`:

```
# ip addr flush dev ge1
```

- Добавление адреса `192.168.0.9` с маской `/24` на интерфейс `ge1`:

```
# ip addr add dev ge1 192.168.0.9/24
```

- Выключение интерфейса `ge1`:

```
# ip link set dev ge1 down
```

- Включение интерфейса `ge1`:

```
# ip link set dev ge1 up
```

- Установка MAC-адреса на `ge1`:

```
# ip link set dev ge1 address 22:22:22:22:22:22
```

Настройки не сохраняются после перезагрузки!