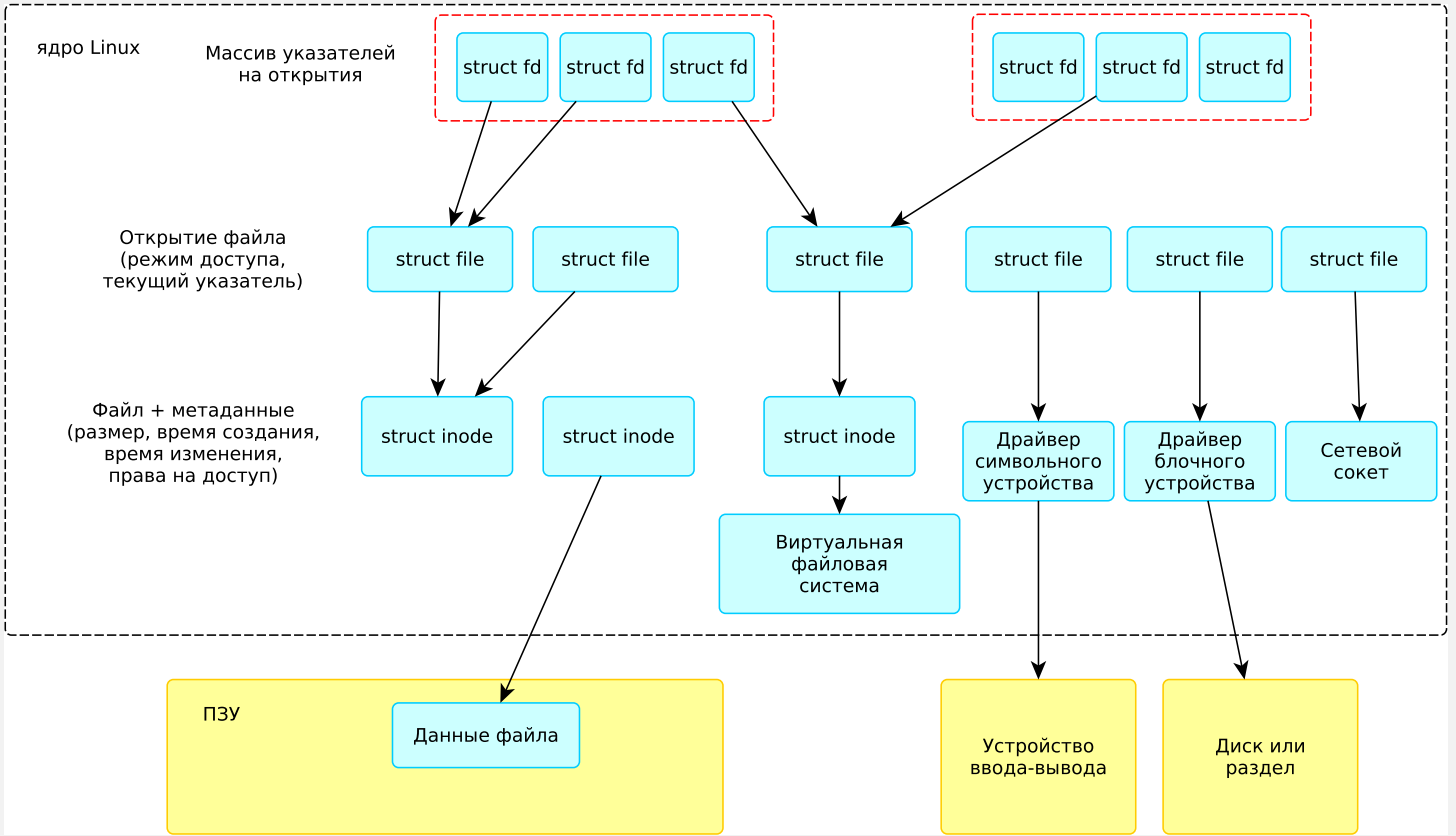
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

12.04.19

Виртуальные файлы



Файлы блочных и символьных устройств

```
# ls -l /dev
brw-rw---- 1 root disk  31,   0 Jan  2 00:54 mtdblock0
crw--w---- 1 root tty    4,  64 Jan  2 00:54 ttyS0
```

- Блочные и символьные файлы не хранят данные. Запросы на чтение и запись исполняет драйвер соответствующего устройства.
- Блочные и символьные файлы - не само устройство, а только интерфейс к нему.
- Соответствие между файлом и драйвером задается старшими и младшими номерами устройства ([31, 0] и [4, 64]). Список существующих устройств можно узнать из файла /proc/devices.
- Файлы можно создавать вручную:

  ```
  # mknod myfile c 4 64
  ```

- Обычно создаются автоматически ядром или приложением udev.

Использование блочных и символьных устройств

- Символьные файлы представляют устройства, передающие поток данных. Пример: терминал /dev/ttyUSB0.
- Символьные файлы не поддерживают произвольный доступ (флаги `O_TRUNC`, `O_APPEND`, функция `lseek`).
- Блочные файлы представляют устройства с произвольным доступом к данным. Обычно это диски и разделы (/dev/sda, /dev/sda1, /dev/mtdblock3).
- На ПК диски имеют имена вида sdX, где X (a, b и т.д.) идентифицирует устройство.
- Разделы имеют имена вида sdXY, где Y - номер раздела.
- На встроенной системе разделы ПЗУ имеют имена вида mtdblockX, где X - номер раздела.

Файловые системы

- ПЗУ - физический диск, состоящий из логических разделов. На каждом разделе - файловая система (FAT, NTFS, ext4, jffs2).
- Несколько файловых систем объединяются в единое дерево файлов.
- Каждая файловая система подключается (монтируется) к определенной точке единого дерева (например, к / или к /home/student).
- Команда для монтирования ФС:

```
# mount <устройство_монтируемого_раздела> <точка_монтирования>
```

- Команда mount без аргументов выводит список примонтированных ФС.
- ФС может быть виртуальной, т.е реализованной в ядре без ПЗУ. Примеры: proc, sysfs, devtmpfs.

Виртуальные файловые системы (proc и devtmpfs)

- Не хранят пользовательские данные. Содержимое может динамически меняться. Данные не сохраняются при перезагрузке.
- Первоначальная роль proc - предоставление информации о процессах. `ps`, `top` получают данные из proc. Обычно монтируется в /proc.
- Вывод списка файловых дескрипторов процесса <PID>:

```
# ls -l /proc/<PID>/fd
```

- Сейчас proc также предоставляет общую информацию о системе. Примеры: список устройств (/proc/devices), список разделов (/proc/partitions).
- devtmpfs содержит символьные и блочные файлы, создаваемые ядром автоматически появлении устройств. Обычно монтируется в /dev.

Представление дерева устройств в sysfs

- Каждое устройство представлено в виде каталога в sysfs.
- Устройства имеют двойную классификацию - по шине (PCI, USB) и по функциональному назначению (последовательный порт, Ethernet-интерфейс).
- Вывод списка всех Ethernet-интерфейсов:

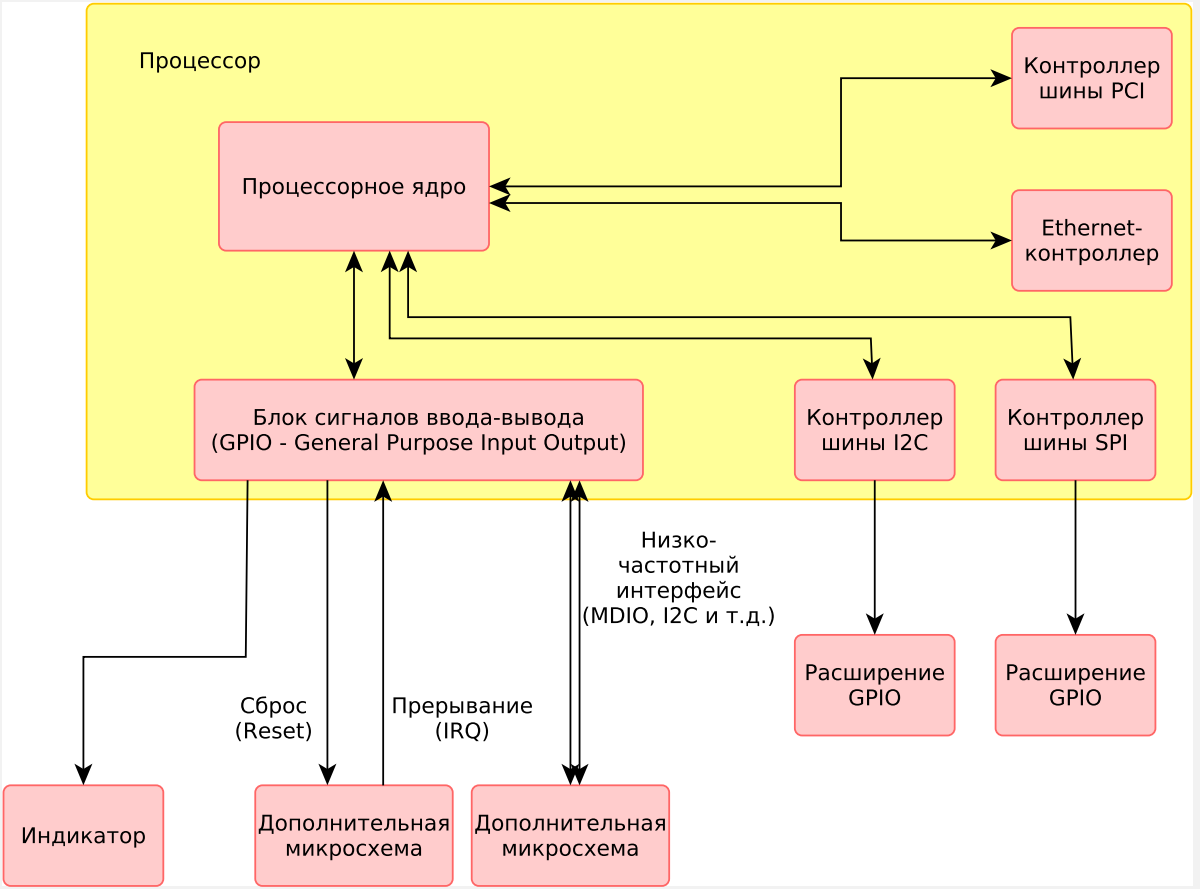
```
# ls /sys/class/net
```

- Вывод списка всех устройств на шине I2C:

```
# ls /sys/bus/i2c/devices
```

- Устройство может иметь читаемые или записываемые файлы (атрибуты). Доступ реализован в драйвере устройства.
- Данные в атрибутах генерируются динамически. Кэширование (т.е. `fopen`, `fread` и т.д.) использовать нельзя!

Сигналы ввода-вывода (GPIO)



Использование GPIO через /sys/class/gpio

- Каждый вывод GPIO имеет номер, назначаемый ядром. Номер сам по себе не имеет физического смысла.
- Каждому аппаратному блоку выводов GPIO соответствует каталог gpiochipXXX, где XXX - номер первого вывода в блоке.
- Атрибуты gpiochip: base - номер первого вывода, ngpio - количество выводов в блоке.
- Перед использованием вывод GPIO (например, номер 472) должен быть экспортирован:

```
# cd /sys/class/gpio
# echo 472 > export
# ls
export       gpio472      gpiochip472  gpiochip480  unexport
```

- Чтение текущего состояния и установка высокого уровня на выводе 472:

```
# cat gpio472/value
# echo 1 > gpio472/value
```

Время в Linux

- Время отсчитывает аппаратный таймер. Этим таймером управляет ядро Linux.
- Приложение может задержать свое исполнение системным вызовом `usleep`

```
int usleep(useconds_t usec);
```

- Выход из функции `usleep` означает завершение запрошенной паузы.
- `usleep` - пример блокирующего вызова, т.е вызова, который приостанавливает (блокирует) приложение до получения результата.
- По умолчанию вызовы `read` и `write` являются блокирующими, т.е. если нет данных для чтения, `read` будет ждать их появления.

Упражнения

- Написать программу для чтения и вывода в терминал текущего состояния вывода GPIO. Номер GPIO задается единственным позиционным аргументом.
- Написать программу для установки значения вывода GPIO. Номер GPIO задается первым позиционным аргументом. Новое значение - вторым.
- Написать программу, которая 4 раза меняет значение GPIO на противоположное с интервалом в 1 секунду. Номер GPIO задается единственным позиционным аргументом.