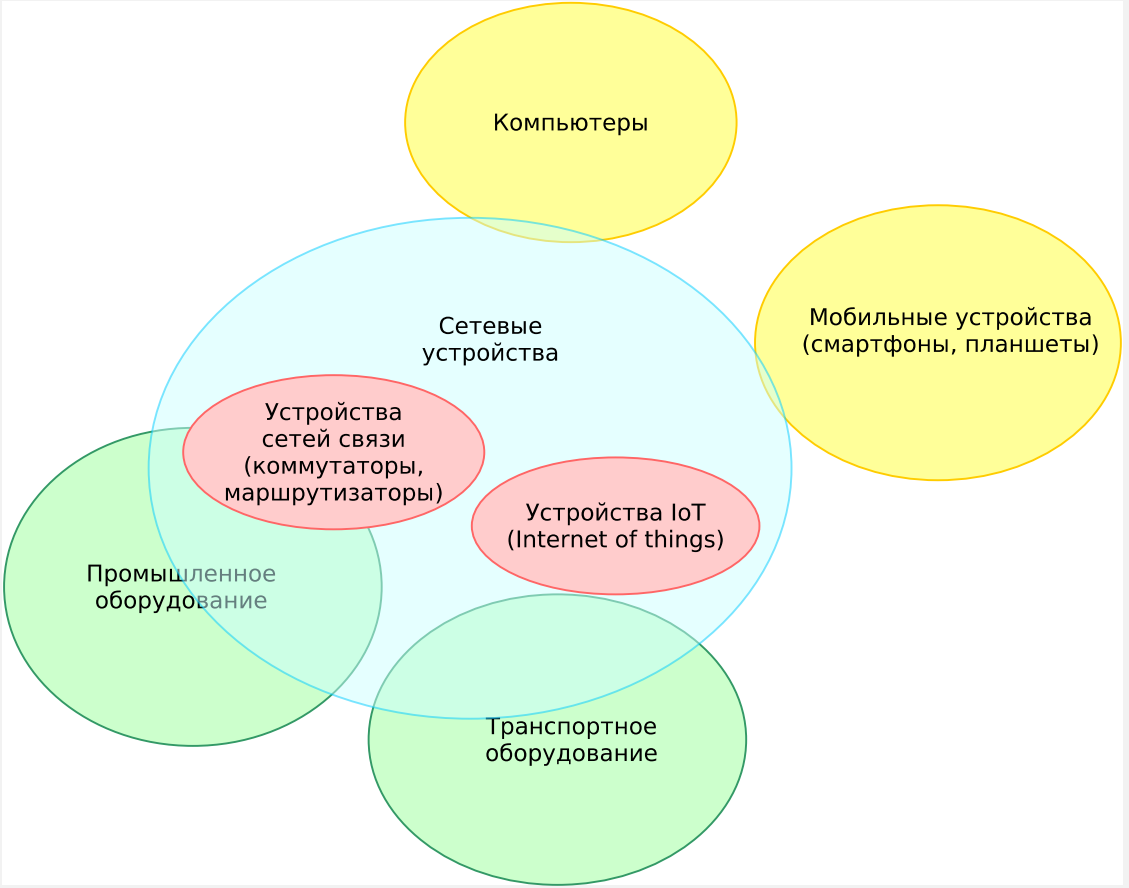
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

26.03.19

Так что такое сетевое устройство

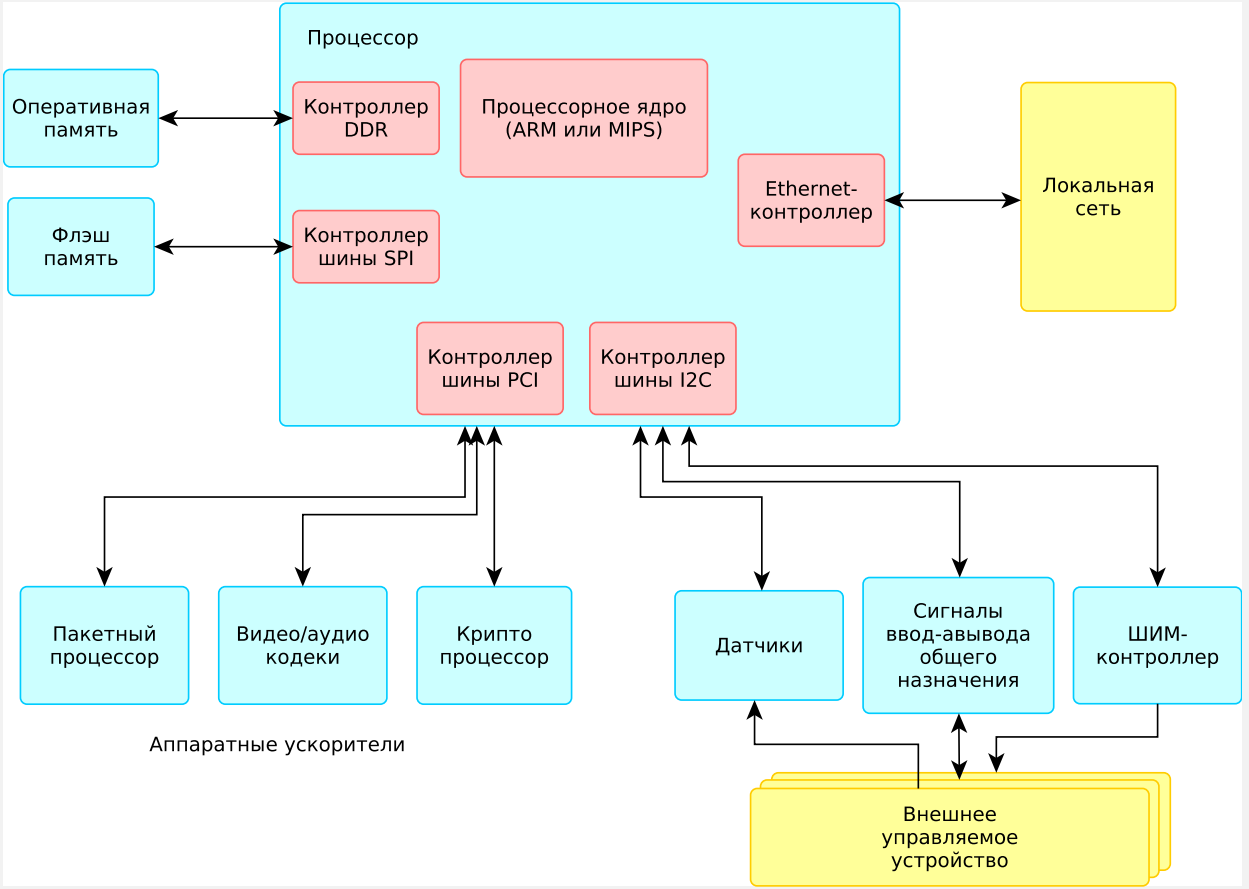


Сетевое устройство как встраиваемая система

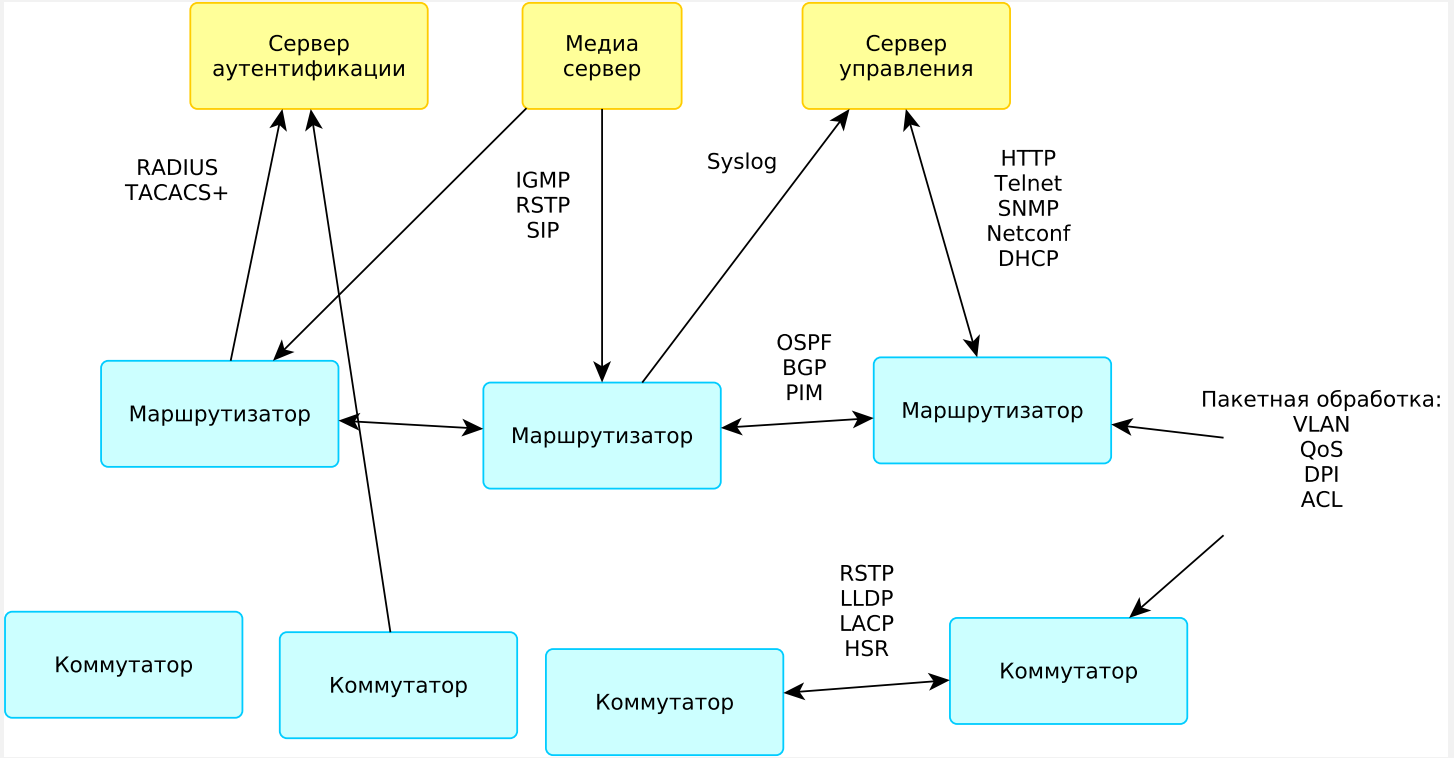
Аппаратное ускорение	Специальные функции
Передача пакетов	Управление дистанционным питанием
Кодирование голоса и видео	Сбор данных с датчиков
[3] Шифрование	Управление промышленными объектами (насосы, двигатели и др.)

В любом случае устройство должно быть эффективно по экономическим показателям (цена самого устройства, цена электричества, охлаждения и т.д.)!

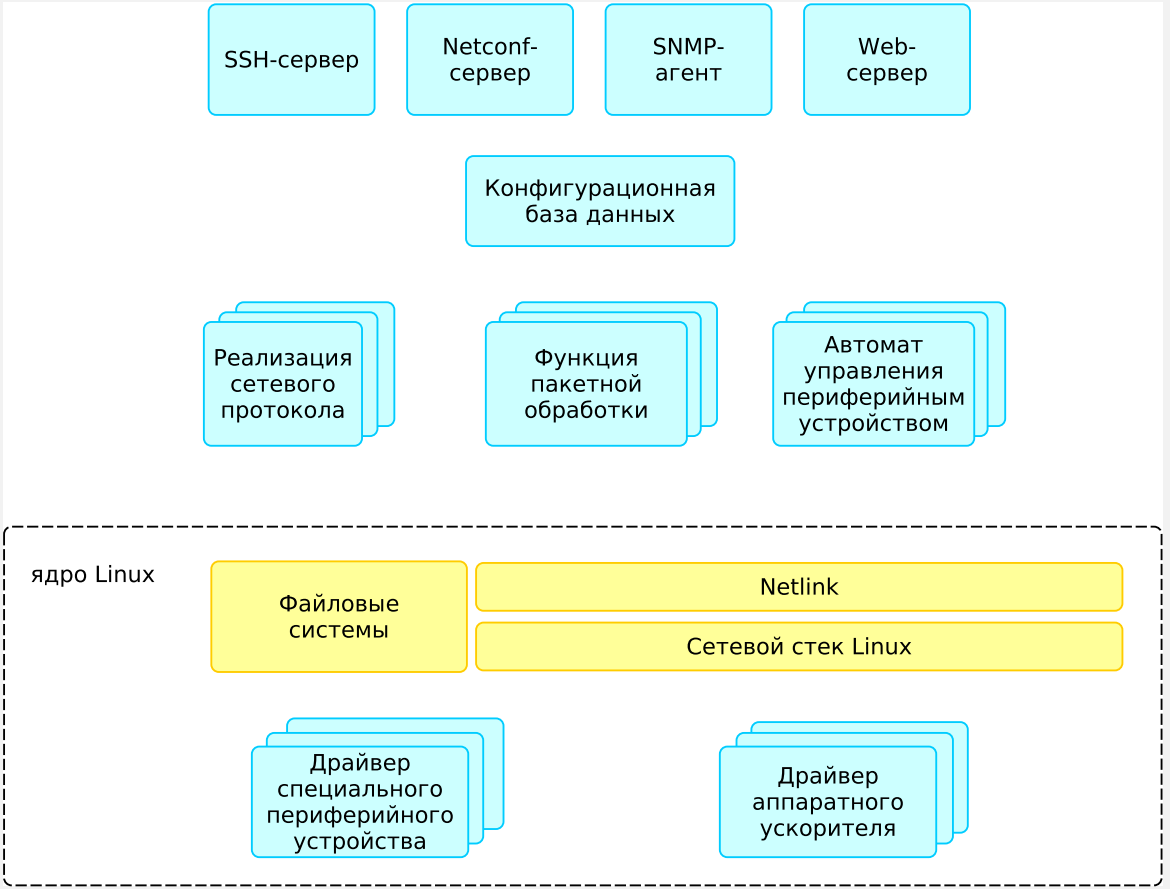
Аппаратная архитектура сетевого устройства



Сетевой аспект



Программная архитектура сетевого устройства



О чем курс?

- **Не** об этом.
  Языки программирования, администрирование коммутаторов Cisco.
- Не столько об этом, но и об этом тоже.
  Linux, программирование на C, сетевые протоколы и технологии.
- Скорее об этом.
  Архитектура ПО сетевого устройства, разработка компонентов ПО СУ, взаимодействие между компонентами, автоматическое управление сетевыми устройствами.

Командный интерпретатор ОС Linux (shell)

shell - интерпретатор + исполнитель скриптов.
shell ~язык программирования (работа с командами и переменными).

- Вывод списка файлов в текущем каталоге (аргумент `-l` запрашивает вывод метаданных).

```
$ ls
$ ls -l
```

- Задание переменной `var`. **Важно!** Пробелов вокруг = быть не должно.

```
$ var=123
```

- Вывод значения переменной `PATH` с помощью команды `echo`.

```
$ echo $PATH
```

- Вывод справки по команде `ls`.

```
$ man ls
```

Дерево файлов в ОС Linux

- Пример абсолютного пути (начинается с /).
  /home/student/file1.txt
- Пример относительного пути (. - текущий каталог).
  ./file1.txt
- Команда вывода текущего каталога.

```
$ pwd
```

- Команда смены текущего каталога.

```
$ cd /home/student/newdir
```

- Команды создание и удаление файла file2 в текущем каталоге.

```
$ touch file2
$ rm file2
```

Синтаксис команды

Локальная установка переменной (только для данной команды)
Путь к команде (относительный путь ищется в каталогах $PATH)
Аргументы команды (передаются в виде массива строк в приложение)
Собственный синтаксис shell

- Запуск программы `myprogram` с заданием переменной окружения VAR, заданием именованных аргументов `a` и `b` и заданием одного позиционного аргумента со значением `posargument`.
  ```
  VAR=123 myprogram -a 1 -b 2 posargument
  ```
- Проверка существования файла `file1` и его вывод в случае успеха.
  ```
  if [ -f file1 ]; then cat file1 ; fi
  ```
- Как узнать, где находится команда `myprogram`.

  ```
  $ which myprogram
  ```

Сборка приложения с помощью GCC

- Простейшая команда сборки приложения hello из исходного файла hello.c. Заголовочные файлы автоматически ищутся в /usr/include, библиотеки автоматически ищутся в /usr/lib.

```
$ gcc -o hello hello.c
```

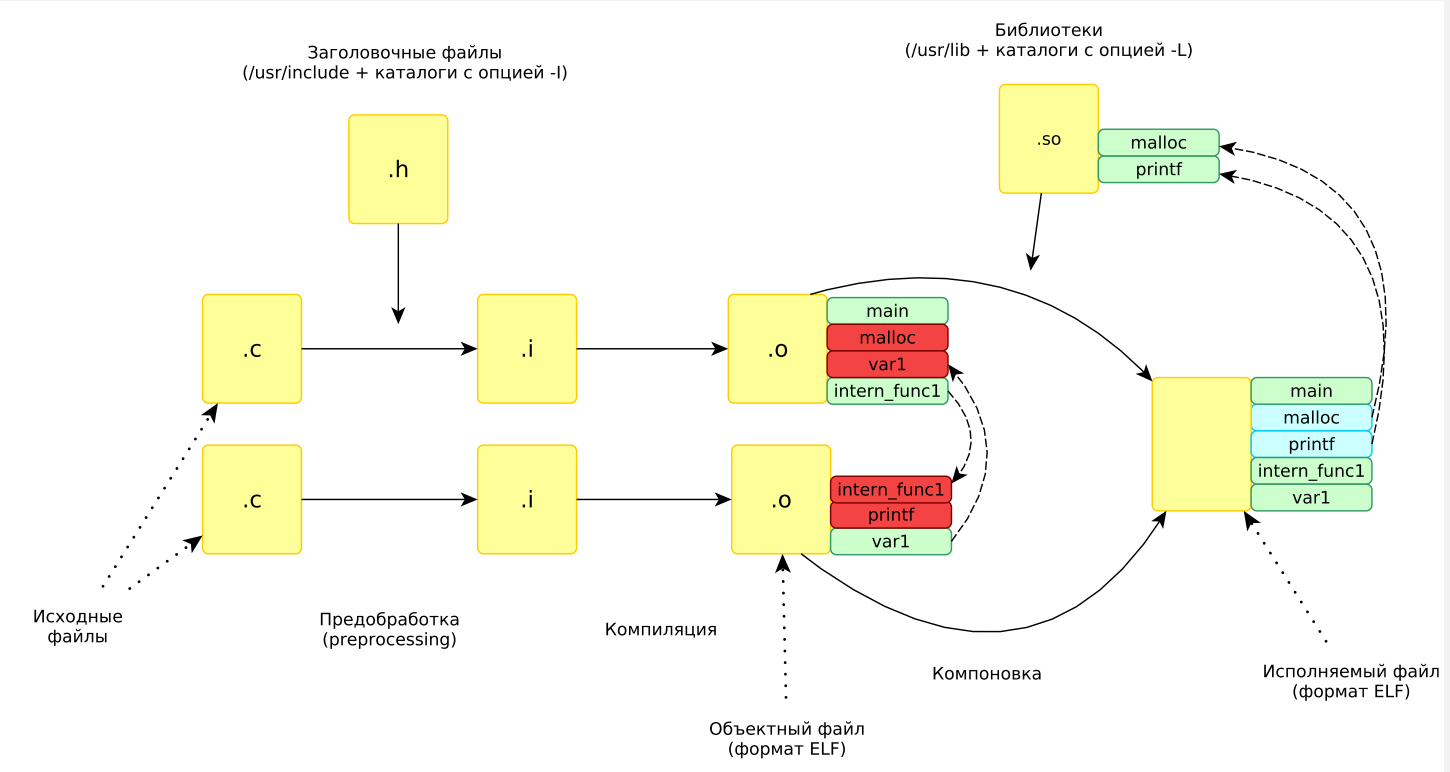
- Сборка с подключением библиотеки libreadline и дополнительными каталогами поиска заголовочных файлов и библиотек.

```
$ gcc -o hello -I./myheaders -L./mylibs -lreadline hello.c
```

- Поэтапная сборка. Сначала исходный файл компилируется в объектный. Потом объектный файл компонуется в приложение.

```
$ gcc -o hello.o -c hello.c
$ gcc -o hello hello.o
```

Этапы сборки



Системы сборки

Системы сборки: make, automake, cmake.
Зачем они нужны?

- Пересборка только для изменившихся файлов.
- Конфигурирование сборки (включение/выключение частей приложения и др.).
- Формирование списка аргументов gcc.

Сборка цели hello с помощью системы сборки make:

```
$ make hello
```

Удаление всех сгенерированных файлов (цель clean):

```
$ make clean
```

Синтаксис Makefile

Цели
Зависимости
Генератор цели
Прочий синтаксис Makefile

```
CFLAGS = -O2
hello: hello.o
        gcc -o $@ $<
hello.o: hello.c
        gcc $(CFLAGS) -c -o $@ $<
```

@ - текущая цель.
< - первая зависимость текущей цели.

Кросс-компиляция (проблемы)

- Другой набор процессорных инструкций.
- Другие заголовочные файлы (из-за отличного набора библиотек, из-за отличий в версиях ядра и библиотек).
- Другие библиотеки.
- Исполняемый файл запускается не на компьютере, а на встроенной системе.

Кросс-компиляция (решение)

- Для сборки используется кросс-компилятор mipsel-oe-linux-musl-gcc (и другие утилиты с таким же префиксом). Компилятор находится в /usr/angtel-sdk/sysroots/x86_64-oesdk-linux/usr/bin/mipsel-oe-linux-musl/.
- Системные библиотеки и заголовочные файлы **не** используются.
- Библиотеки и заголовочные файлы подключаются из /usr/angtel-sdk/sysroots/mip и /usr/angtel-sdk/sysroots/mips32el-nf-oe-linux-musl/usr/include, соответственно. Для этого у компилятора GCC настраивается „виртуальный корень” (sysroot) со значением /usr/angtel-sdk/sysroots/mips32el-nf-oe-linux-musl.
- В Makefile добавляется цель install, которая записывает исполняемый файл на плату по протоколу FTP.

Подключение коммутатора „Корунд”

Ethernet-порт

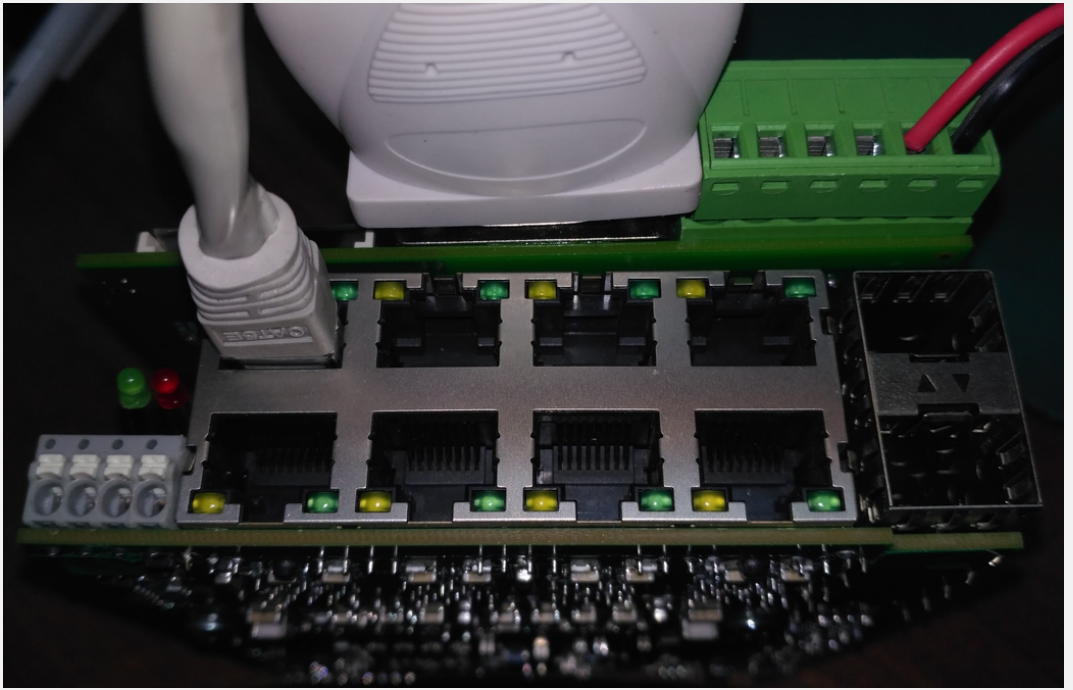
Проверка связи:
ping 192.168.0.8

Подключение:
telnet 192.168.0.8

Последовательный порт
(UART, RS232)

Подключение:
minicom -D /dev/ttyUSB0

Питание 24-48 В



Работа с FTP-клиентом

- Подключение к устройству:

```
$ ftp 192.168.0.8
```

- имя пользователя - root, пароль - пустая строка.
- Загрузка файла file1 на плату:

```
ftp> put file1
```

- Скачивание файла file2 с платы:

```
ftp> get file2
```

- Завершение FTP-сессии:

```
ftp> quit
```

Работа с Telnet-клиентом

- Подключение к устройству:

```
$ telnet 192.168.0.8
```

- Вывод информации о системе:

```
# uname -a
```

- Разрешение исполнения файла file1:

```
# chmod +x file1
```

- Запуск приложения file1:

```
# ./file1
```

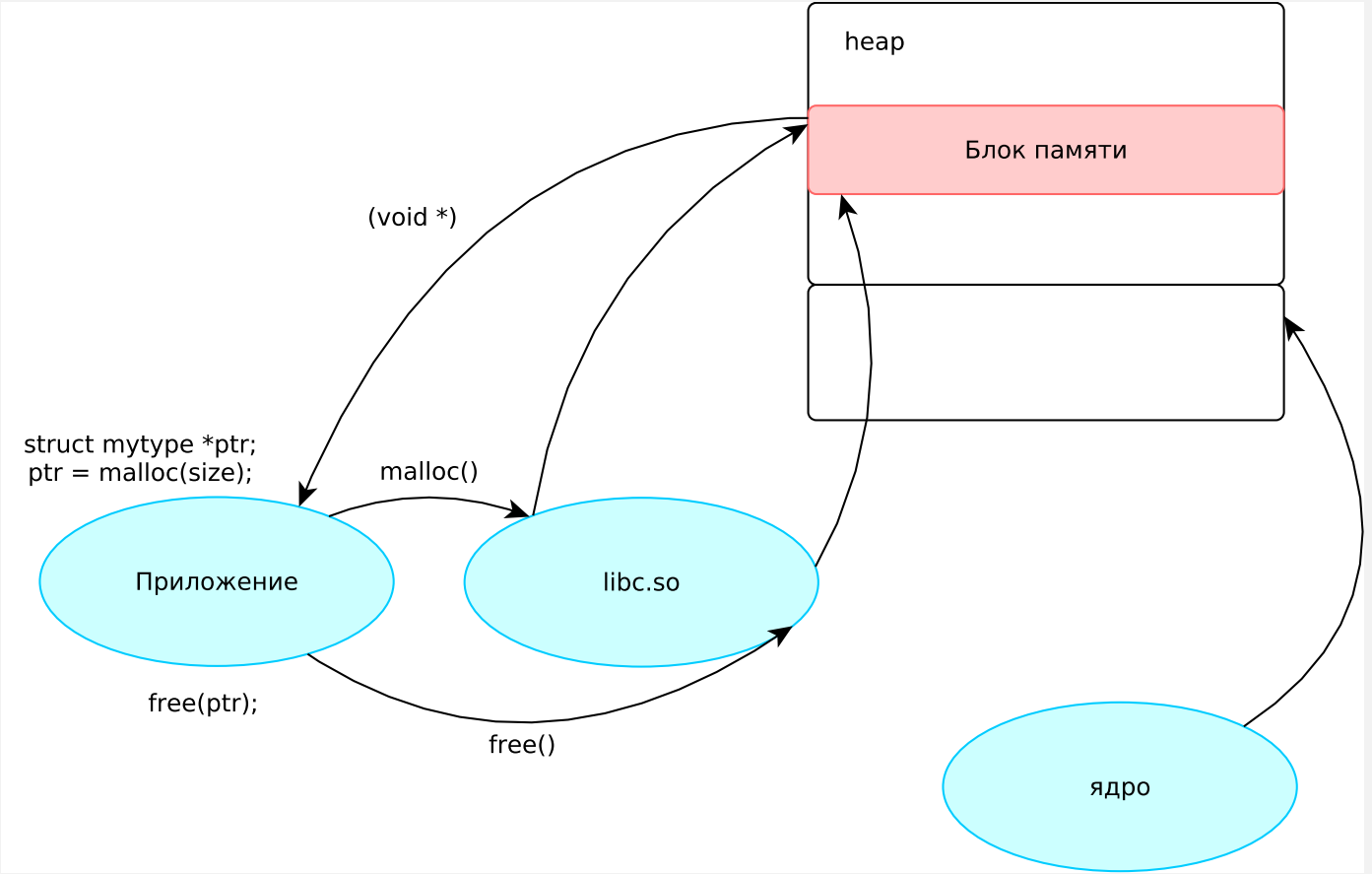
- Завершение сессии:

```
# exit
```

C-строки

- Представление строки „text” в языке C: [’t’, ’e’, ’x’, ’t’, ’\0’].
- Длина строки `str` (без нулевого байта): `strlen(str)`.
- Сравнение строк `s1` и `s2` (результат 0, если равны): `strcmp(s1, s2)`.
- Вывод форматированной строки в буфер `buf` размера `size`: `snprinf(buf, size, fmt, ...)`.
- Преобразование строки `str` в число: `atoi(str)`.

Динамическое выделение памяти



Задание

Как должна выглядеть рабочая программа:

```
# ./prog
2 + 5 -    3
Result: 4
#
```

Чтение строки из потока ввода выполняется с помощью функции `getline`. Если строка еще не введена, `getline будет ожидать ввода. После завершения вызова возвращается строка в динамически выделенном буфере.`