Практика 1. Воспоминания про Linux. Воспоминания про получение двоичного кода (компиляция, ассемблер, линковка).

Евгений Линский

Воспоминания про Linux

Что и как установить?

- Что? Ubuntu (самая распространенная)
- Как?
  - второй ОС (установка сложнее, быстро работает)
  - в виртуальной машине
- Что почитать? Основы Linux от основателя Gentoo

Что это?

```
/
bin -> usr/bin
boot
cdrom
dev
etc
home
lib -> usr/lib
lib32 -> usr/lib32
lib64 -> usr/lib64
libx32 -> usr/libx32
lost+found
media
mnt
opt
proc
root
run
sbin -> usr/sbin
snap
srv
swapfile
sys
tmp
usr
var
```

FHS

/ (корневая директория)

- /boot (статичные файлы загрузчика)
- /dev (файлы устройств)
- /etc (специфические для хоста конфигурационные файлы)
- /home (домашняя директория пользователя)
- /lib (основные разделяемые библиотеки и модули ядра)
- /mnt (точка монтирования для временных нужд)
- /opt (дополнительные пакеты ПО)
- /sbin (основные системные программы)
- /tmp (временные файлы)
- /usr (вторичная иерархия)
- /var (изменяемые данные)

Командная строка

- gnome-terminal (программа для ввода и вывода команд)
- bash (программа для выполнения команд)
- команды для работы с файловой системой
  - cd dir (перейти в директорию), ls dir (посмотреть содержание директории)
  - cp src dst (скопировать файл), mv src dst (переместить файл), rm dir (удалить файл)
  - mkdir dir (создать директорию)

bash: tab (автодополнение), arrows (предыдущая команда), Ctrl+R (поиск по истории)

Пути

- Пусть в домашней директории пользователя elinsky есть директории caos и soac.
- В директории caos есть файл hello.S
- Мы находимся в директории soac.
- Как можно указать путь до файла hello.S

Пути

- Пусть в домашней директории пользователя elinsky есть директории caos и soac.
- В директории caos есть файл hello.S
- Мы находимся в директории soac.
- Как можно указать путь до файла hello.S
- ls /home/elinsky/caos/hello.S

Пути

- Пусть в домашней директории пользователя elinsky есть директории caos и soac.
- В директории caos есть файл hello.S
- Мы находимся в директории soac.
- Как можно указать путь до файла hello.S
- ls /home/elinsky/caos/hello.S
- ls ../caos/hello.S

Пути

- Пусть в домашней директории пользователя elinsky есть директории caos и soac.
- В директории caos есть файл hello.S
- Мы находимся в директории soac.
- Как можно указать путь до файла hello.S
- ls /home/elinsky/caos/hello.S
- ls ../caos/hello.S
- ls ~/caos/hello.S

Примеры

```
cd ~
mkdir caos
mkdir soac
cd caos
nano hello.S
cat hello.S
cp hello.S ../caos
rm hello.S
man rm
```

- nano – консольный текстовый редактор (графический gedit)
- cat – утилита для просмотра содержимого файла
- man – документация по утилите rm

Примеры. Ключи командой строки.

```
cd ~
mkdir -p ~/a/c/d
ls -al
mkdir -p ~/x
cp -r ~/a ~/x
rm -rf ~/x
ls -al
```

Как установить программу?

Собрать самому (make, зависимости)/установить пакет (репозитории).

```
sudo apt update
sudo apt search gcc
sudo apt install build-essential gcc-multilib
```

gcc

- gcc – набор компиляторов
- binutils – линкер, ассемблер, дизассемблер
- gdb – отладчик

К следующей паре

Хорошо бы:

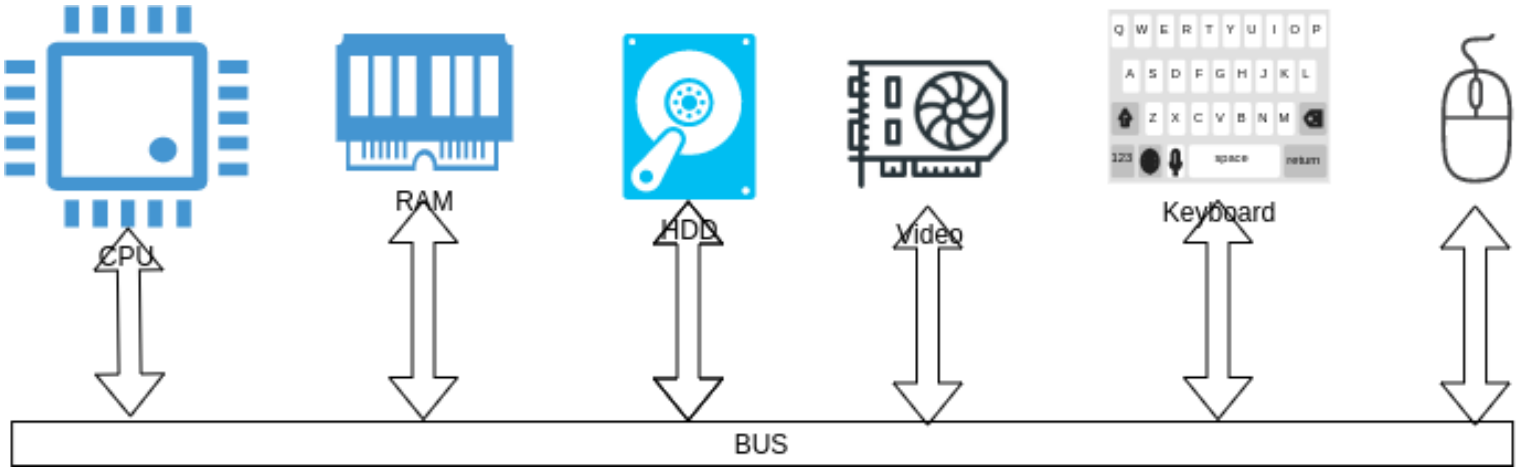
- поставить Ubuntu (можно в виртуалке)
- поставить build-essential (gcc и все-все-все)

Воспоминания про двоичный код

Все хранится в числах

- Компьютер умеет работать только с числами
  - Текст. Кодировка задает соответствие между изображением символа и числовым кодом ('A' – 65).
  - Изображение. Цвет точки на экране — три числа Red Green Blue (черный — 0 0 0).
  - Команды, из которых состоит программа, тоже хранится в виде чисел.
- Числа хранятся в двоичной системе счисления
  - "Есть сигнал/нет сигнала" (1/0) [ложь! ложь! ложь!]
  - "Есть намагниченность/нет намагниченности" (1/0)

Схема компьютера



Можно сказать, что все что умеет процессор, это выполнять над числами арифметические и логические операции и пересылать данные между периферийными устройствами:

- арифметическая операция: загрузить числа из памяти в процессор, выполнить операции, выгрузить в память
- вывести пиксель: переслать координаты и цвет точки (RGB) в видеокарту
- сохранить файл: послать данные и их положение на диске в контроллер жесткого диска

Программирование в двоичных кодах

- Архитектура процессора (x86, ARM, RISC-V): набор команд и регистры
  - Регистры — ячейки памяти внутри процессора (x86: размер – 64 бита, количество – 20)
- Программа

загрузить из ячейки RAM 300 в регистр1	1 300 1
загрузить из ячейки RAM 500 в регистр2	1 500 2
сложить регистр1 и регистр2 в регистр3	3 1 2 3
выгрузить регистр3 в ячейку RAM 100	2 3 100

- 1 – код команды загрузить, 2 – код команды выгрузить, 3 – ... сложить

Проблемы при программировании в кодах

- Надо помнить о ячейках памяти и регистрах (какие заняты, какие нет)
- Коды команд плохо запоминаются
- Отсутствует переносимость: в разных архитектурах – разные наборы команды, коды команд, наборы регистров

Ассемблер

- Ассемблер – транслятор (программа) из текста на языке ассемблер в двоичные коды.
- У команд и ячеек памяти есть символьные имена, которые легко запомнить.
- Программа:

загрузить из ячейки RAM 300 в регистр1	load data1, r1
загрузить из ячейки RAM 500 в регистр2	load data2, r2
сложить регистр1 и регистр2 в регистр3	add r1, r2, r3
выгрузить регистр3 в ячейку RAM 100	store r3, data3
	.data data1 300 data2 500
	.data data3 100

Какие проблемы остались?

Ассемблер

- Ассемблер – транслятор (программа) из текста на языке ассемблер в двоичные коды.
- У команд и ячеек памяти есть символьные имена, которые легко запомнить.
- Программа:

загрузить из ячейки RAM 300 в регистр1	load data1, r1
загрузить из ячейки RAM 500 в регистр2	load data2, r2
сложить регистр1 и регистр2 в регистр3	add r1, r2, r3
выгрузить регистр3 в ячейку RAM 100	store r3, data3
	.data data1 300 data2 500
	.data data3 100

Какие проблемы остались? Переносимость!

Язык Высокого Уровня (ЯВУ)

```
1  int a = 3;
2  int b = 5;
3  int c = a + b;
```

- Компилятор – транслятор (программа) из текста на ЯВУ переводит в текст на языке ассемблера.
- Подбирает команды, ячейки памяти и номера регистров так, чтобы программа быстро выполнялась и занимала минимум памяти (за несколько проходов по тексту).

Программа –> [Компилятор] –> [Ассемблер] –> Исполняемый файл

Язык C

- Bell Labs. Задача: создать переносимую ОС (должна работать на разных архитектурах).
- Язык для ОС Unix. Язык C (“Write once, compile everywhere!”). Деннис Ритчи. 197X.
- Если для новой архитектуры существует (в современном мире — обычно да) компилятор языка C, то программу не надо переписывать, а надо просто перекомпилировать этим компилятором.

Язык C

- Bell Labs. Задача: создать переносимую ОС (должна работать на разных архитектурах).
- Язык для ОС Unix. Язык C (“Write once, compile everywhere!”). Деннис Ритчи. 197X.
- Если для новой архитектуры существует (в современном мире — обычно да) компилятор языка C, то программу не надо переписывать, а надо просто перекомпилировать этим компилятором.

На самом деле нет: в стандартную библиотеку языка C не входит графика, сеть и т.д.

Файлы

```
1  //main.c
2  int main() {
3    int a = 3; int b = 5;
4    int c = a + b;
5    int d = sum(c, b);
6    return 0;
7  }
```

```
1  //util.c
2  int sum(int a, int b) {
3    return a + b;
4  }
```

Построение программы (build)

- Скомпилировать (компилятор + ассемблер) main.c –> main.o (объектный файл)
- Скомпилировать util.c –> util.o
- Слинковать [Линкер] main.o, util.o –> main.elf

Линкер

main.o

```
1 300 1
1 500 2
3 1 2 3
2 3 100
call sum
...
```

sum.o

```
func sum
43 500 2
4 1 2 3
4 3 100
...
```

В объектном файле – имена функций не заменены на адреса.

Линкер

main.elf

```
[20] 1 300 1
[24] 1 500 2
[28] 3 1 2 3
[32] 2 3 100
[36] 13  50
...
[50] 43 500 2
[54] 4 1 2 3
[58] 4 3 100
...
```

Линкер (линковщик, компоновщик) должен склеить файлы вместе и заменить вызовы функции по имени на вызовы по адресу (процессор понимает только числа – адреса).

Библиотеки.

1. Статические (*.a, *.lib): объектные файла из библиотеки присоединяются к программе в момент линковки
   - Легко установить (не нужно отдельно загружать библиотеку)
   - При выходе новой версии библиотеки (например, исправлен баг) автору программы нужно послать пользователю пересобранную версию
2. Динамические (*.so, *.dll): хранятся отдельно, связывание программы и библиотеки происходит в момент выполнения (помогает отдельная компонента — загрузчик)
   - Необходимо отдельно установить все необходимые библиотеки (решение: packages and repositories)
   - При выходе новой версии библиотеки (например, исправлен баг) пользователь может самостоятельно обновить только библиотеку без пересборки программы.

gcc

```
// собрать полностью с отладочными символамм
gcc -g hello.c -o hello.elf
// только компиляция и ассемблирование (без линковки)
gcc -c hello.c -o hello.o
// только компиляция
gcc -S hello.c -o hello.S
// только препроцессор
gcc -E hello.c -o hello.tmp
// дизассемблирование
objdump -D hello.elf >hello.bin
```

Запуск и отладка

```
./hello.elf
echo $? // результат return из main
gdb hello.elf
```

PATH – путь для поиска исполняемых файлов (echo $PATH)

GDB

```
layout src
break main
run
s
p/d varibale
layout asm
si
continue
quit
```

Ctrl + x + o

Makefile

оступы – tab!

```
hello.elf: hello.c
  gcc hello.c -o hello.elf

clean:
  rm -rf hello.elf

build: hello.elf
```

$> make

К следующей паре

Хорошо бы:

- позапускать gcc в разных режимах
- позапускать gdb в разных режимах