

Есть три операции, которые вы можете производить с файлом:

↓
read write execute

ls -l file.txt показывает ваши полномочия. Вот как интерпретировать выведенные данные:

rw-	rw-	r--	bork staff
↑	↑	↑	
bork	staff	ОСТАЛЬНЫЕ	
(пользователь) может читать и записывать файл	(группа пользователей) может читать и записывать файл	любой может читать файл	

Права доступа к файлам составляют 12 бит

setuid setgid
↓ ↓
OOO
↑
sticky
для пользователя для группы пользователей для всех
rwX rwX rwX

У файлов:

r = read (чтение)
w = write (запись)
x = execute (выполнение)

У каталогов это соответственно:

r = показать содержимое
w = создавать, менять и удалять файлы и папки
x = переходить по директориям (cd) и изменять файлы

110 в двоичной системе счисления равняется 6

rw-	r--	r--
= 110	= 100	= 100
= 6	= 4	= 4

chmod 644 file.txt означает изменение прав доступа на rw-, r--, r--.

Всё просто!

Setuid влияет на исполняемые файлы

\$ls -l /bin/ping

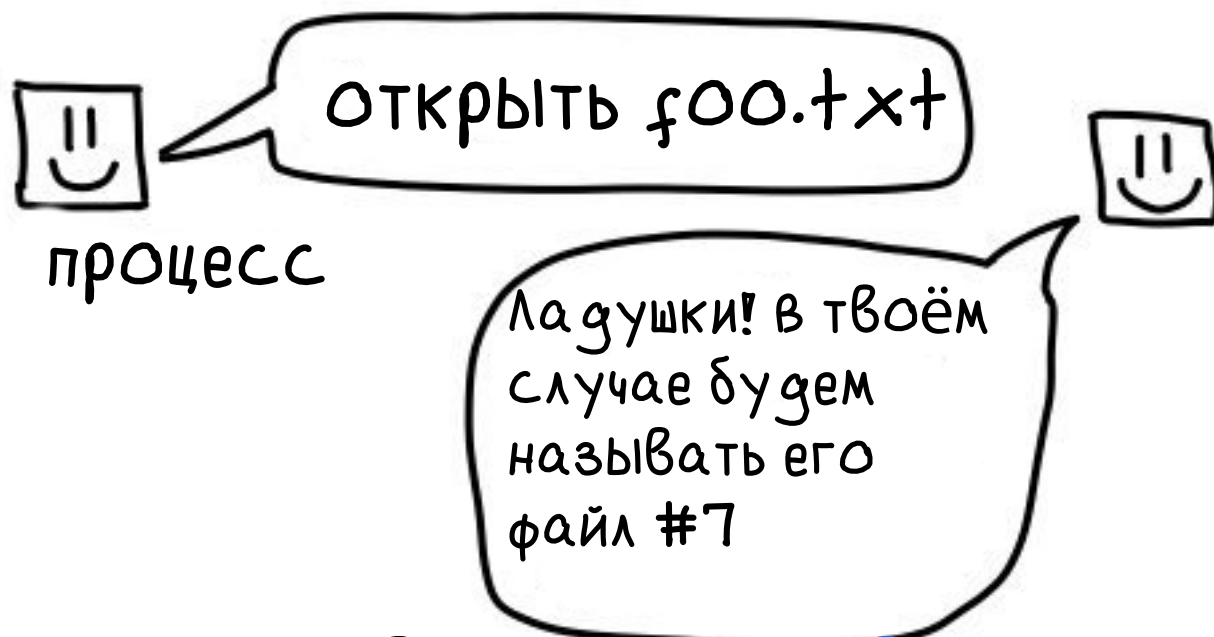
rwS r-x r-x root root
↑
означает, что пинг всегда запускается с правами root

Setgid выполняет три разные, не связанные между собой, действия для исполняемых файлов, каталогов и обычных файлов



Файловые дескрипторы

Системы Unix используют целые числа, чтобы отслеживать открываемые файлы



Их называют **файловыми дескрипторами**

lsOf (**l**ist **o**pen **f**iles) покажет открытые файлы процесса.

```
$ lsOf -p 4242 ← PID, который нас интересует
```

FD	ИМЯ
0	/dev/pts/tty1
1	/dev/pts/tty1
2	pipe: 29174
3	/home/julia/origennaya.txt
5	/tmp/

↑
FD значит файловый дескриптор

Файловые дескрипторы могут применяться к:

- файлам на диске
- программным каналам
- сокетам (сетевым соединениям)
- терминалам (например, xterm)
- устройствам (твоей колонке! /dev/null!)
- ЕЩЕ МНОГО ЧЕМУ (eventfd, inotify, signalfd, epoll и т.д., и т.д.)



Когда вы открываете или редактируете файл, программный канал или сетевое соединение, вы делаете это, используя файловый дескриптор

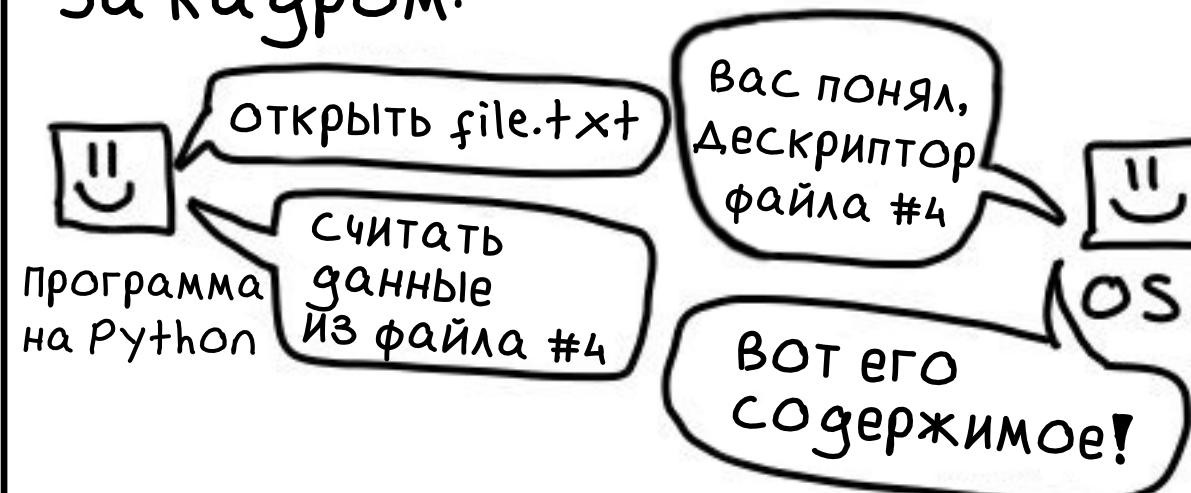


Давайте посмотрим под капот. Вот как работает один простой код на Python:

Python:

```
f = open("file.txt")  
f.readlines()
```

За кадром:



Каждый процесс (нууу... почти каждый) обладает тремя стандартными файловыми дескрипторами:

```
stdin    → 0  
stdout   → 1  
stderr   → 2
```

"read from stdin" означает "считать данные из файлового дескриптора 0"

↑
может также являться программным каналом, файлом или терминалом

ПОТРЯСАЮЩИЙ КАТАЛОГ: /proc

Джулия Эванс
@b0rk

У каждого процесса на Linux есть PID (Process Identification Number, идентификационный номер процесса), например, 42.

В /proc/42, находится ОЧЕНЬ МНОГО полезной информации о процессе 42

/proc/PID/cmdline

аргументы командной строки, с которыми был запущен процесс

/proc/PID/exe

символическая ссылка на бинарник процесса.

МАГИЯ: ссылка работает, даже если бинарник был удалён!

/proc/PID/environ

Все переменные окружения для процесса

/proc/PID/status

Программа запущена или приостановлена? Сколько памяти она использует? И много другой полезной информации!

/proc/PID/fd

каталог со всеми файлами, которые открывались процессом! запустите `$ ls -l /proc/42/fd`, чтобы увидеть все файлы для процесса 42.

Эти симлинки тоже заряжены волшебством, поэтому их можно использовать для восстановления удалённых файлов ♥

/proc/PID/stack

текущий стек ядра. Бывает полезно, если процесс завис в системном вызове

/proc/PID/maps

карты распределения памяти процесса. Библиотеки общего пользования, динамическая память, анонимные карты и т.д.

и ещё



загляните в

man proc

и узнаете больше!

Как ★ эффективно★ общаться, когда ты не согласен

Джулия Эванс
@bork



Лайфхак: ♥ прояви любопытство ♥

Когда я с чем-то не согласна, я спрашиваю собеседника о его собственном опыте! Часто оказывается, что он делал что-то такое, что я не пробовала (а бывает и наоборот). В таком случае иногда получается узнать что-то новое



Лайфхак: постарайтесь не говорить, что собеседник неправ. Укажите на то, что неверны его убеждения. НЕТ: «ты не прав» ДА: «это неправильно». Мы все стремимся к одному — к истине! ♥

RR – отладчик, который позволит тебе вернуться в прошлое!



Должно быть, его трудно использовать, верно?

Ну, по сути, это то же самое, что использование gdb!



Вот как его применять:

```
$ rr record /your/application --args
```

...

Что-то пошло не так!!

Давай посмотрим, что именно...

```
$ rr replay  
GNU gdb (gdb) ...
```

В итоге имеем сессию gdb, содержащую:

- ★ Те же самые результаты системных вызовов
- ★ Те же самые адресные пространства
- ★ реверс-версии многих gdb-команд

reverse-continue reverse-finish

reverse-next reverse-step

Попробуй использовать это вместо gdb!

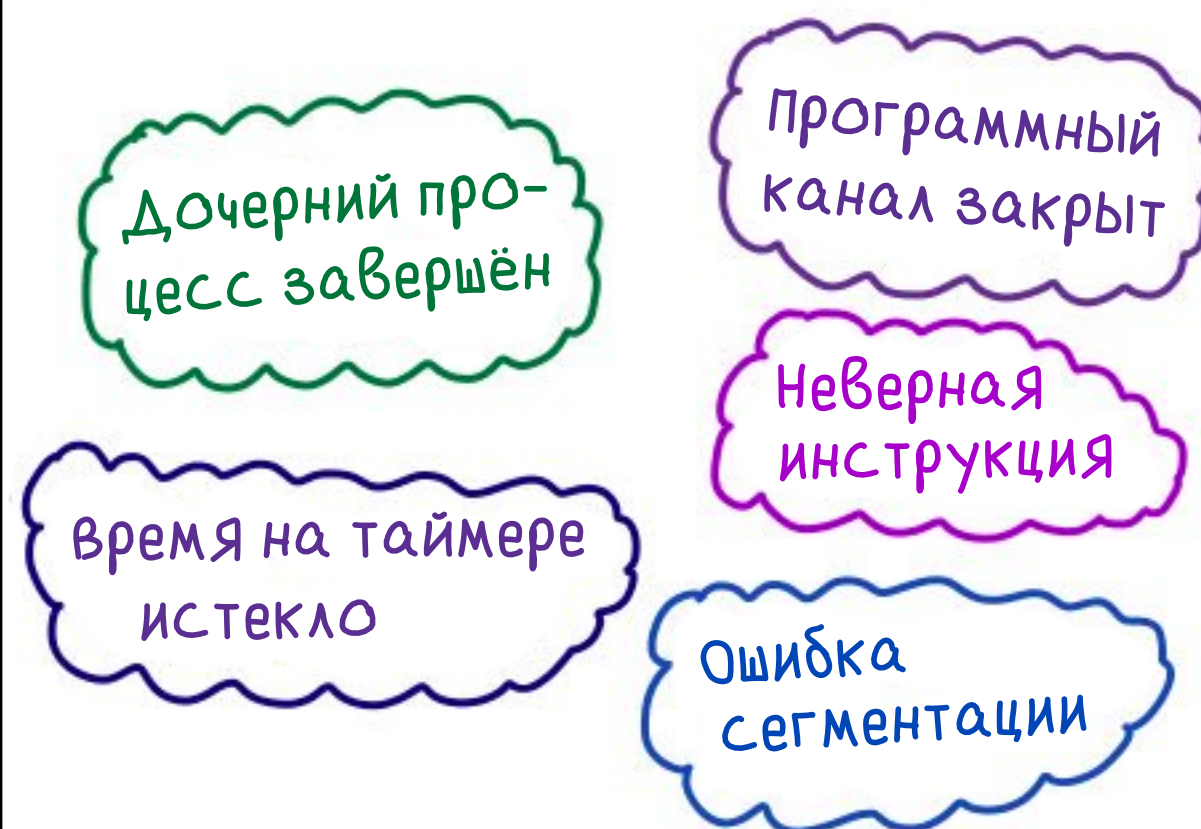
СИГНАЛЫ

drawings.jvns.ca

Если ты когда-нибудь
запускал **kill**
значит, ты использовал
сигналы



Ядро Linux посылает
вашему процессу сигналы
во многих ситуациях



Вы сами можете посылать
сигналы тем же системным
вызовом **kill** или одной из
приводимых ниже команд:

SIGINT `ctrl-c`
SIGTERM `kill`
SIGKILL `kill -9` } разные
уровни
«смерти»

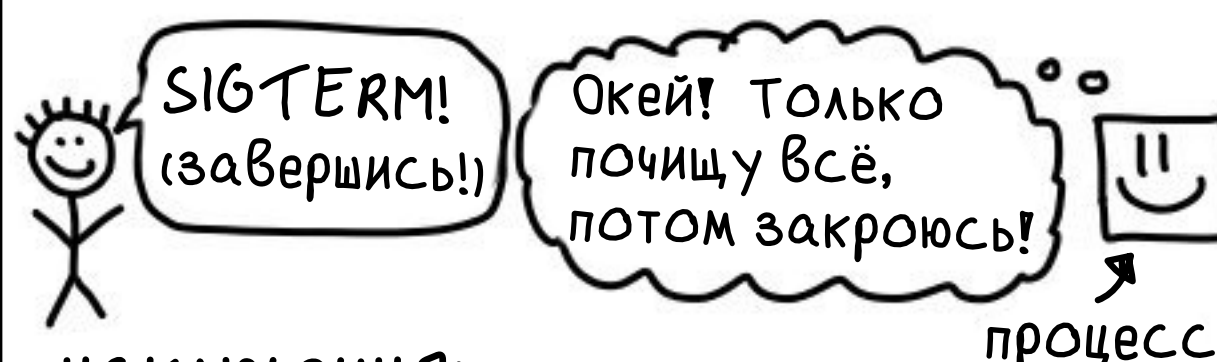
SIGHUP `kill -HUP`

↑
Часто интерпретируется как
«перезагрузить конфигурацию»
(`reload config`), например, в `nginx`

Каждый сигнал выполняет
одно из стандартных действий

- игнорировать
- уничтожить процесс
- уничтожить процесс и
создать файл с содержимым
рабочей памяти процесса
- остановить процесс
- восстановить процесс

Твоя программа может
использовать специальные
обработчики практически
для любого сигнала



Исключения:

SIGSTOP и SIGKILL

не могут быть проигнорированы
на него
кастанули
SIGKILL



Обработка сигналов может
стать непростой задачей,
потому что в ЛЮБОЙ момент
может появиться новый сигнал



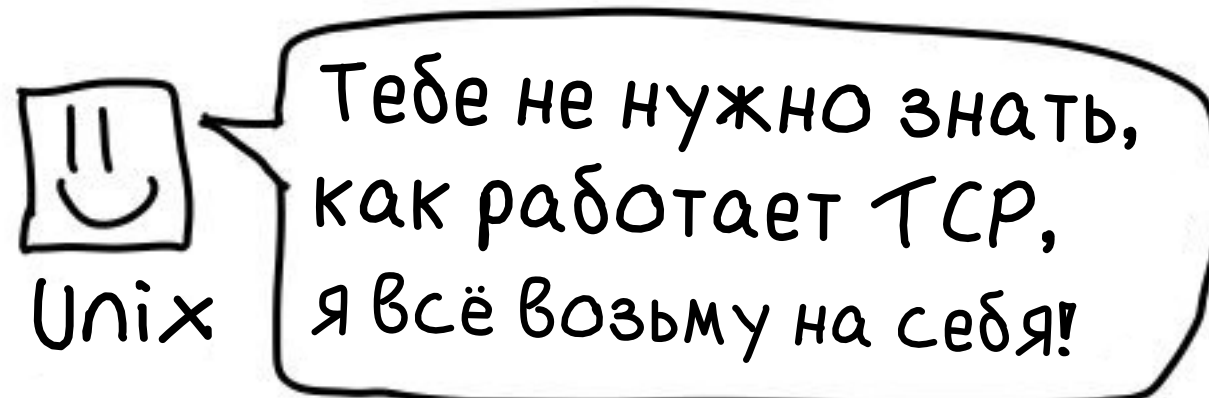
СОКЕТЫ

drawings.jvns.ca

Сетевые протоколы
зачастую сложны



Unix системы наделены программным интерфейсом приложения, который называется «Socket API». Он упрощает создание сетевых соединений (на Windows тоже!))

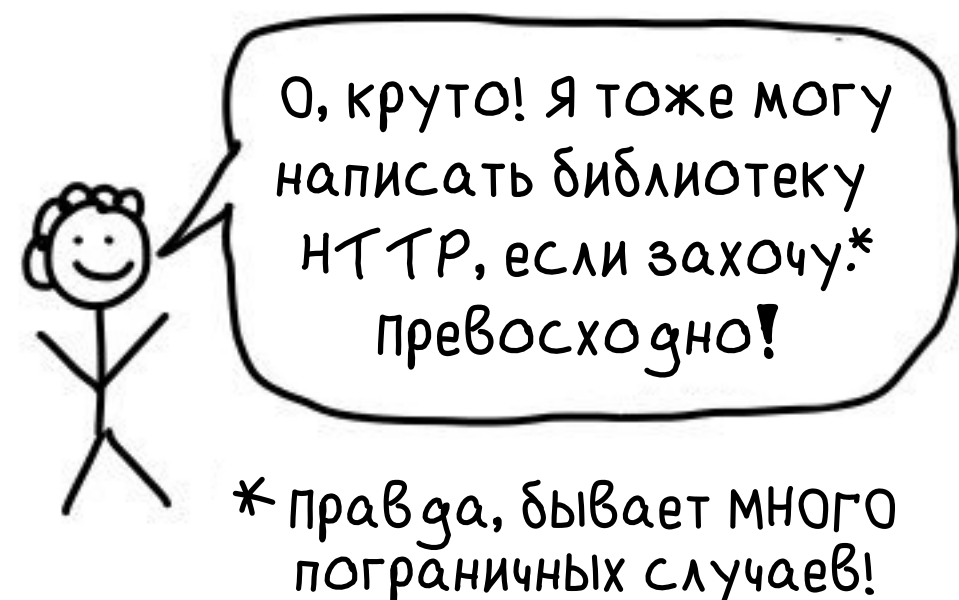


Вот как выглядит процесс загрузки фотографии с котиками при помощи Socket API:

- ① Создайте сокет
`fd=socket(AF_INET, SOCK_STREAM ...)`
- ② Подключитесь к IP/порту
`connect(fd, 12.13.14.15:80)`
- ③ Сделайте запрос
`write(fd, "GET /cat.png HTTP/1.1 ...)`
- ④ Получите ответ
`cat-picture=read(fd...`

Если посмотреть под капот любой библиотеки HTTP, можно заметить, что везде используются сокет.

`$ curl awesome.com` ← сокет
`Python: requests.get("yay.us")` ↓



AF_INET? Что это?

AF_INET, в общем-то, это и есть «интернет-сокет»: он позволяет устанавливать соединение с другими компьютерами по Интернету по IP'шнику.

Его главной альтернативой является AF_UNIX («сокет домена Unix»), служащий для установления соединения между программами на одном компьютере.

Три вида Интернет сокетов (AF_INET):

SOCK_STREAM = TCP
это использует curl

SOCK_DGRAM = UDP
это использует dig (DNS)

SOCK_RAW = просто дайте мне отправить IP-пакеты, и я выполню свой собственный протокол
это использует ping

Сокеты домена Unix

Джулия Эванс
@b0rk

Сокеты домена Unix — это файлы.

```
$ file mysock.sock  
socket
```

Права доступа к файлу определяют, кто может отправлять данные этому сокету.

Они позволяют двум программам на одном компьютере взаимодействовать друг с другом.

Например, сокеты домена Unix используются Docker'ом!



Существует два вида сокетов домена Unix:

stream как TCP!
позволяет отправлять непрерывный поток байтов

datagram как UDP!
Отправляет дискретные фрагменты данных.



Преимущество 1

Позволяет тебе работать с правами доступа к файлу, чтобы ограничить доступ к HTTP/базам данных!

```
chmod 600 secret.sock
```

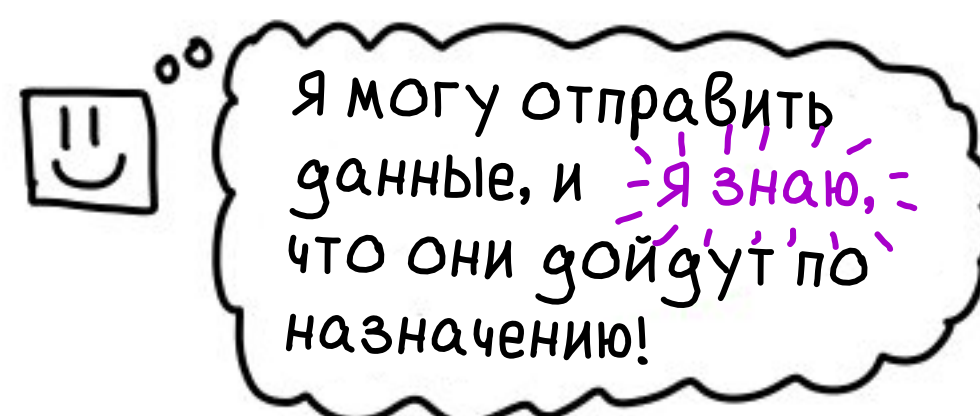
Поэтому Docker использует сокеты домена Unix 🔒



Преимущество 2

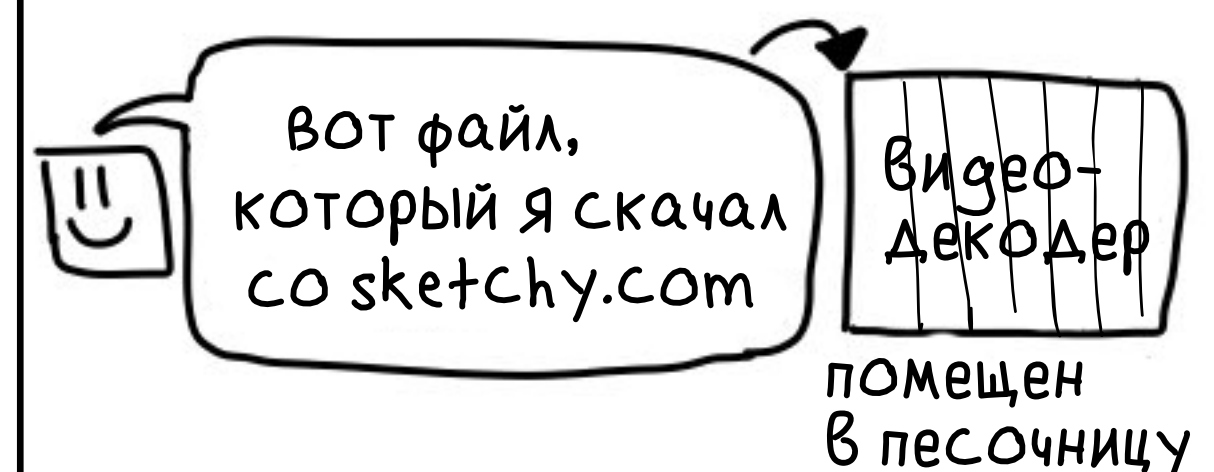
UDP сокеты не всегда надёжны (даже на одном и том же компьютере).

Датаграмный сокет домена Unix всегда надёжен и не подвергается переупорядочиванию!



Преимущество 3

Вы можете отправить файловый дескриптор через сокет домена Unix. Удобно, когда нужно обработать сомнительные входные файлы!



Что такое программные потоки

Джулия Эванс
@b0rk

drawings.jvns.ca

Каждый процесс может
иметь множество
потоков

process id	thread id
1888	1888
1888	1892
1888	1893
1888	2007

Потоки совместно используют память

ПОТОК
1

Я сейчас запишу
"3" по адресу
2977886

А я могу пере-
писать это,
если захочу

ПОТОК
2

Но они могут запускать
разный код

ПОТОК
1

Я провожу
вычисления

А я жду
окончания
сетевого
запроса!

ПОТОК
2

Совместное использование памяти может вызвать
проблему «состояния гонки»



Если у тебя процессор
с 8 ядрами, ты можешь
запускать код для
восьми потоков
ОДНОВРЕМЕННО

1 5
2 6
3 7
4 8

ядро процессора

Мы такие
занятые
☺ ☺



Понравилось?
Найти больше можно здесь!
это бесплатно!
drawings.jvns.ca

Переведено командой FirstVDS

Другие переводы комиксов Джулии тут
<https://firstvds.ru/blog/julia-evans>