Московский Государственный Технический Университет им. Н.Э. Баумана

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

к.т.н.
Папулин Сергей Юрьевич
papulin_bmstu@mail.ru

Москва, 2021

Лекция. Виртуальная файловая система

Файлы

Файлы. Определение

- Файлы являются логическими информационными блоками, создаваемыми процессами

- Файл является механизмом абстрагирования. Он предоставляет способ сохранения информации на диске и последующего ее считывания, который должен оградить пользователя от подробностей о способе и месте хранения информации и деталей фактической работы дисковых устройств

- Когда процесс создает файл, он присваивает ему имя. Когда процесс завершается, файл продолжает существовать, и к нему по этому имени могут обращаться другие процессы

- Файлами управляет операционная система. Структура файлов, их имена, доступ к ним, их использование, защита, реализация и управление ими являются основными вопросами разработки операционных систем

Типы файлов

- **Обычные файлы** – файлы, содержащие информацию пользователя. Как правило, к обычным файлам относятся файлы ASCII и двоичные файлы.

- **Каталоги** – системные файлы, предназначенные для поддержки структуры файловой системы

- **Символьные специальные файлы** имеют отношение к вводу-выводу и используются для моделирования последовательных устройств ввода-вывода, к которым относятся терминалы, принтеры и сети.

- **Блочные специальные файлы** используются для моделирования дисков

Системные вызовы для работы с файлами

Системный вызов	Описание
`fd = creat(name, mode)`	Один из способов создания нового файла
`fd = open(file, flags, mode)`	Открыть файл для чтения, записи либо и того и другого одновременно
`s = close(fd)`	Закрыть открытый файл
`n = read(fd, buffer, nbytes)`	Прочитать данные из файла в буфер
`n = write(fd, buffer, nbytes)`	Записать данные из буфера в файл
`position = lseek(fd, offset, whence)`	Переместить указатель в файле
`s = stat(name, &buf)`	Получить информацию о состоянии файла
`s = fstat(fd, &buf)`	Получить информацию о состоянии файла
`s = pipe(&fd[0])`	Создать канал
`s = fcntl(fd, cmd, ...)`	Манипуляции с файловым дескриптором (блокировка записей, файла и другие операции

Поля stat

- Устройство, на котором располагается файл
- Номер i-узла (идентифицирует файл на устройстве)
- Режим файла (включая информацию о защите)
- Количество ссылок файла
- Идентификатор владельца файла
- Группа, к которой принадлежит файл
- Размер файла в байтах
- Время создания
- Время последнего доступа
- Время последней модификации

Системные вызовы для работы с каталогами

Системный вызов	Описание
`s = mkdir(path, mode)`	Создать новый каталог
`s = rmdir(path)`	Удалить каталог
`s = link(oldpath, newpath)`	Создать ссылку на существующий файл
`s = symlink(oldpath, newpath)`	Создать soft ссылку
`s = unlink(path)`	Удалить ссылку
`s = chdir(path)`	Изменить рабочий каталог
`dir = opendir(path)`	Открыть каталог для чтения
`s = closedir(dir)`	Закрыть каталог
`dirent = readdir(dir)`	Прочитать одну запись каталога
`rewinddir(dir)`	Установить указатель в каталоге на первую запись

inode (1)

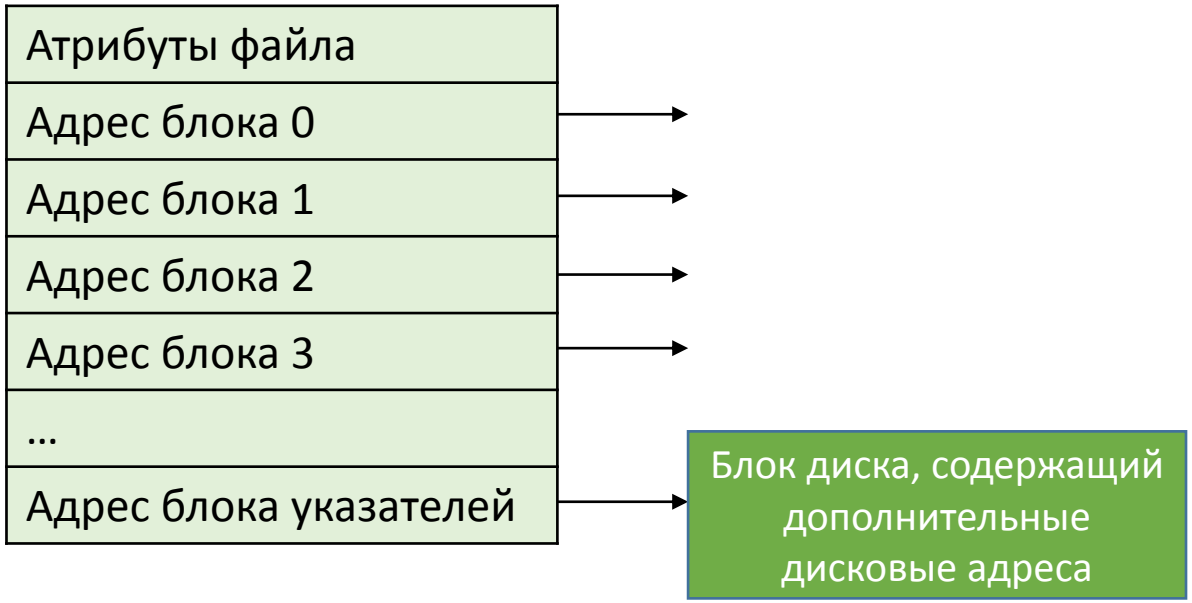
- Важным вопросом при реализации файлового хранилища является отслеживание соответствия файлам блоков на диске

- Одним из методов отслеживания принадлежности конкретного блока конкретному файлу является связь с каждым файлом структуры данных, называемой inode (index-node — индекс-узел), содержащей атрибуты файла и дисковые адреса его блоков

- При использовании inode появляется возможность найти все блоки файла. Большим преимуществом этой схемы является то, что inode должен быть в памяти только в том случае, когда открыт соответствующий файл

inode (2)

– С inode связана одна проблема: если каждый узел имеет пространство для фиксированного количества дисковых адресов, то что произойдет, когда файл перерастет этот лимит? Одно из решений заключается в резервировании последнего дискового адреса не для блока данных, а для блока, содержащего дополнительные адреса блоков



Виртуальная файловая система

Виртуальная файловая система

- Большинство UNIX-систем, в том числе и Linux, пытаясь интегрировать несколько файловых систем в упорядоченную структуру, использовали концепцию виртуальной файловой системы (Virtual File System – VFS)

- При этом разные файловые системы могут быть подключены в виде каталогов корневой файловой системы

- С пользовательской точки зрения это будет единая иерархическая файловая система, поскольку объединение нескольких несовместимых файловых систем невидимо для пользователей или процессов

Архитектура виртуальной файловой системы (1)

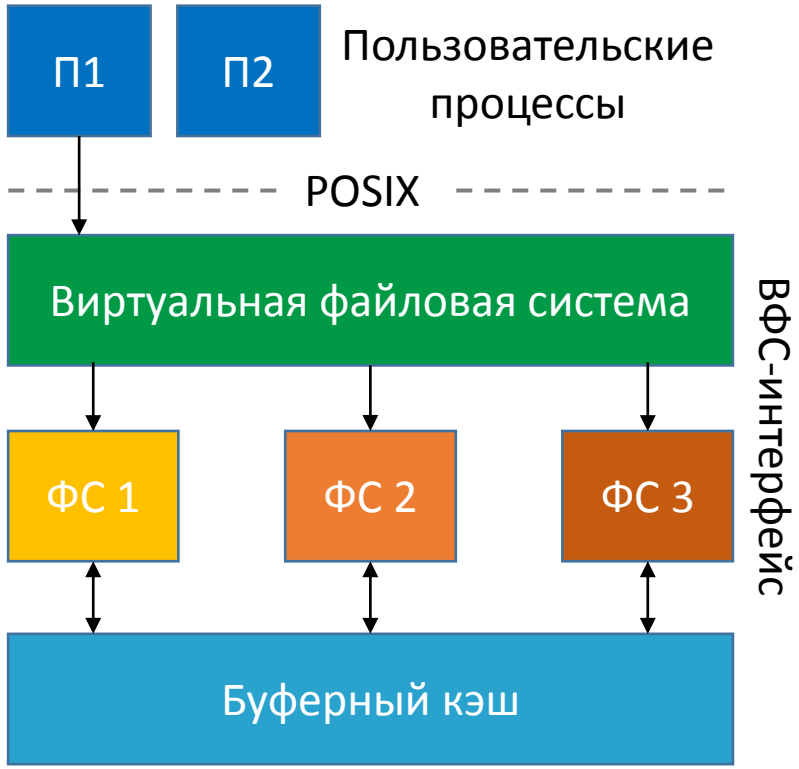
- ВФС имеются два интерфейса: «верхний» — к пользовательским процессам и «нижний» — к конкретным файловым системам.

- Все относящиеся к файлам системные вызовы направляются для первичной обработки в адрес виртуальной файловой системы. Эти вызовы, поступающие от пользовательских процессов, являются стандартными POSIX-вызовами, такими как open, read, write, lseek и т.д. Таким образом, ВФС обладает «верхним» интерфейсом к пользовательским процессам

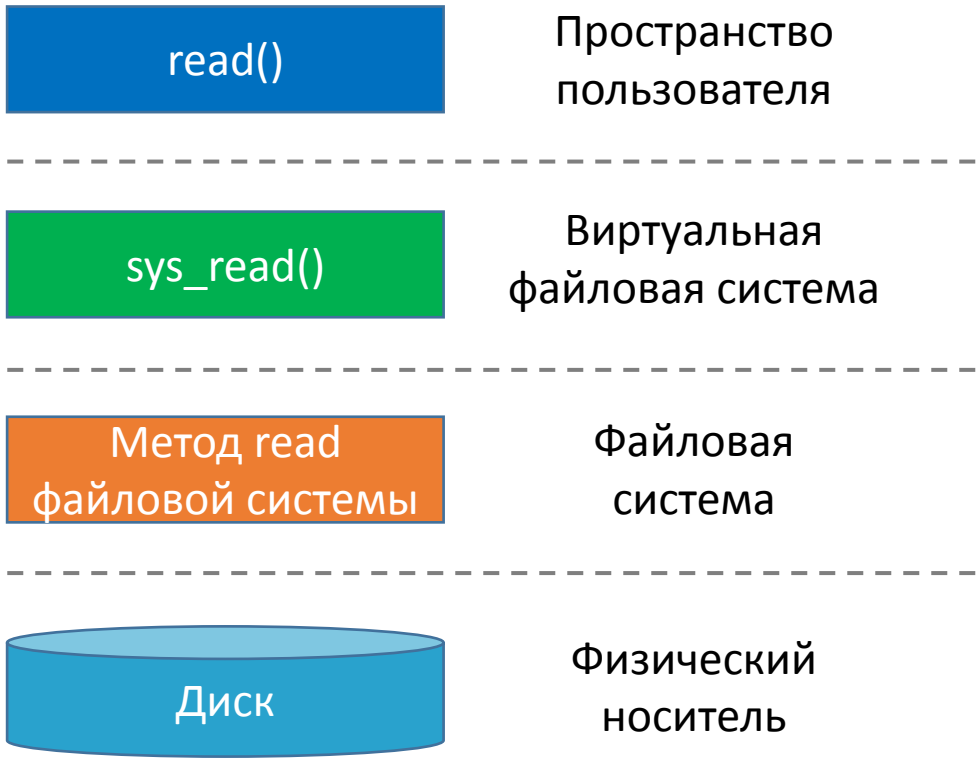
- У ВФС есть также «нижний» интерфейс к конкретной файловой системе – ВФС-интерфейс.

- Этот интерфейс состоит из нескольких десятков вызовов функций, которые ВФС способна направлять к каждой файловой системе для достижения конечного результата, например, считывающая с диска конкретный блок, помещающая его в буферный кэш файловой системы и возвращающая указатель на него.

Архитектура виртуальной файловой системы (2)



Пример системного вызова



Регистрация файловой системы

- При загрузке системы ВФС регистрирует корневую файловую систему (при подключении (монтировании) других файловых систем, либо во время загрузки, либо в процессе работы они также должны быть зарегистрированы в ВФС)

- При регистрации файловой системы она предоставляет список адресов функций, необходимых ВФС, таких как чтение/запись блоков данных

- После установки файловую систему можно использовать

Последовательность действий при открытии файла (1)

Если файловая система была подключена к каталогу /usr и процесс осуществил вызов:

```
open("/usr/include/unistd.h", O_RDONLY)
```

- при анализе пути ВФС увидит, что к /usr была подключена новая файловая система
- определит местоположение ее **суперблока**, просканировав список суперблоков установленных файловых систем
- после этого ВФС может найти **корневой каталог** установленной файловой системы, а в нем — путь include/unistd.h
- Затем ВФС создает **vnode** (в Linux называется **inode**) и направляет вызов конкретной файловой системе, чтобы вернулась вся информация, имеющаяся в **inode** файла

Последовательность действий при открытии файла (2)

- Эта информация копируется в **vnode** (в оперативной памяти) наряду с другой информацией, наиболее важная из которой — указатель на **таблицу функций**, вызываемых для операций над **vnode**, таких как чтение — **read**, запись — **write**, закрытие — **close** и т. д.

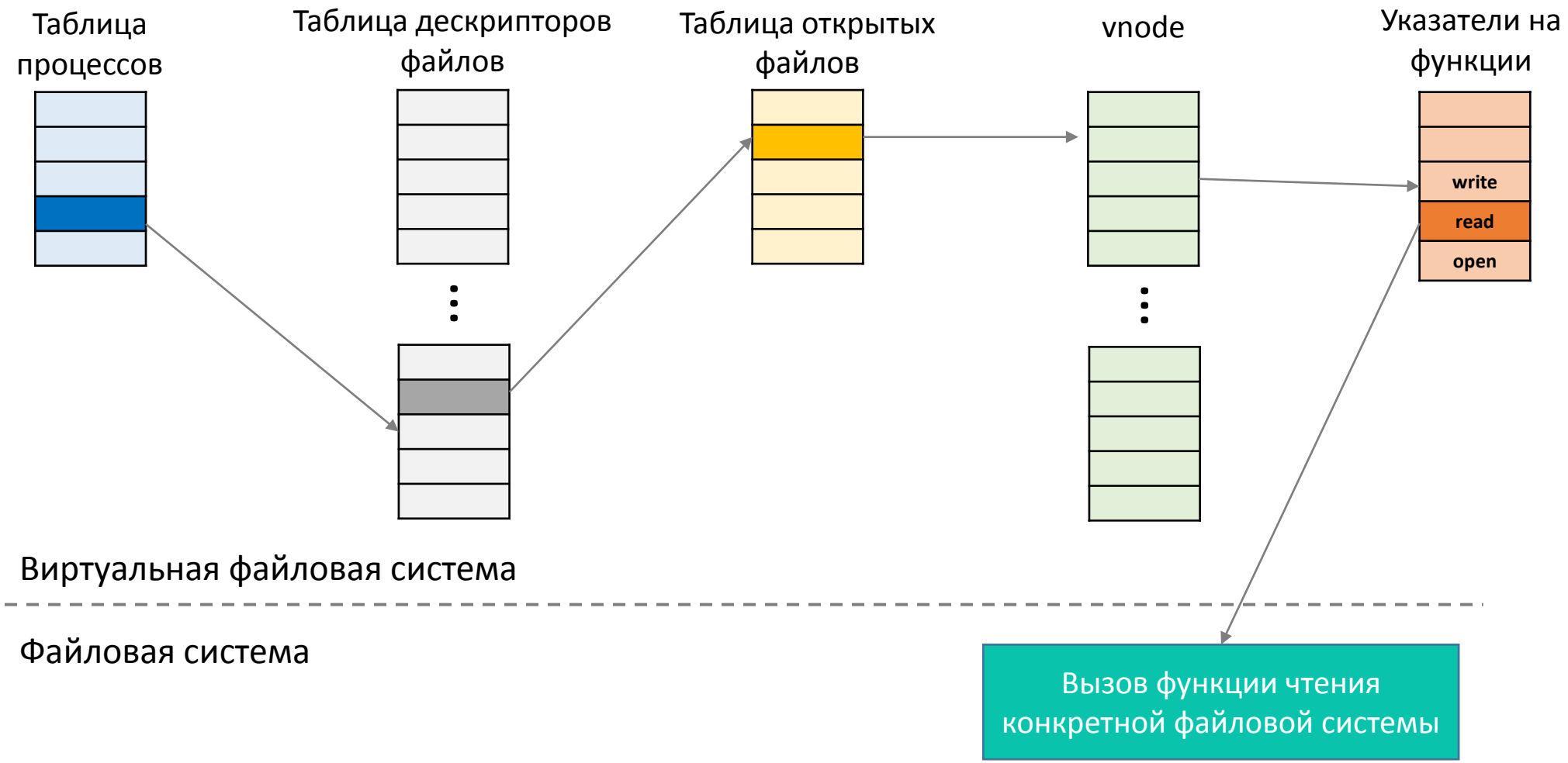
- После создания **vnode** ВФС создает запись в таблице **дескрипторов файлов** вызывающего процесса и настраивает ее так, чтобы она указывала на новый **vnode** (**дескриптор файла** на самом деле указывает на другую структуру данных, в которой содержатся текущая позиция в файле и указатель на **vnode**)

- ВФС возвращает **дескриптор файла** вызывавшему процессу, чтобы тот мог использовать его при чтении, записи и закрытии файла.

- процесс осуществляет чтение, используя **дескриптор файла**, ВФС находит **vnode** из **таблиц процесса** и **дескрипторов файлов** и следует по указателю к **таблице функций**

- вызывается функция, управляющая чтением, и внутри конкретной файловой системы запускается код, извлекающий требуемый блок

Последовательность действий при открытии файла (3)



Виртуальная файловая система в Linux

Объекты

- **Суперблок**
  Конкретная файловая система
  Операции: read_inode, sync_fs
- **Элемент каталога** (dentry)
  Элемент каталога, компонент пути
  Операции: create, link
- **inode**
  Конкретный файл
  Операции: d_compare, d_delete
- **Файл** (file)
  Открытый файл, связанный с процессом
  Операции: read, write

Объекты

- Суперблок содержит критичную информацию о компоновке файловой системы. Разрушение суперблока делает файловую систему нечитаемой.

- inode описывает один файл. В Linux каталоги и устройства также представлены файлами, так что они тоже имеют соответствующие inode. И суперблок, и inode имеют соответствующие структуры на том физическом диске, где находится файловая система

- ВФС поддерживает структуру данных *dentry*, которая представляет элемент каталога. Эта структура данных создается файловой системой на ходу. Управление иерархической структурой файловых систем достигается за счет её использования. Элементы каталога кэшируются в так называемом *dentry_cache*

- Структура данных *file* является представлением открытого файла в памяти, она создается в ответ на системный вызов *open*. Она поддерживает такие операции, как *read*, *write*, *sendfile*, *lock* и пр.

Объекты

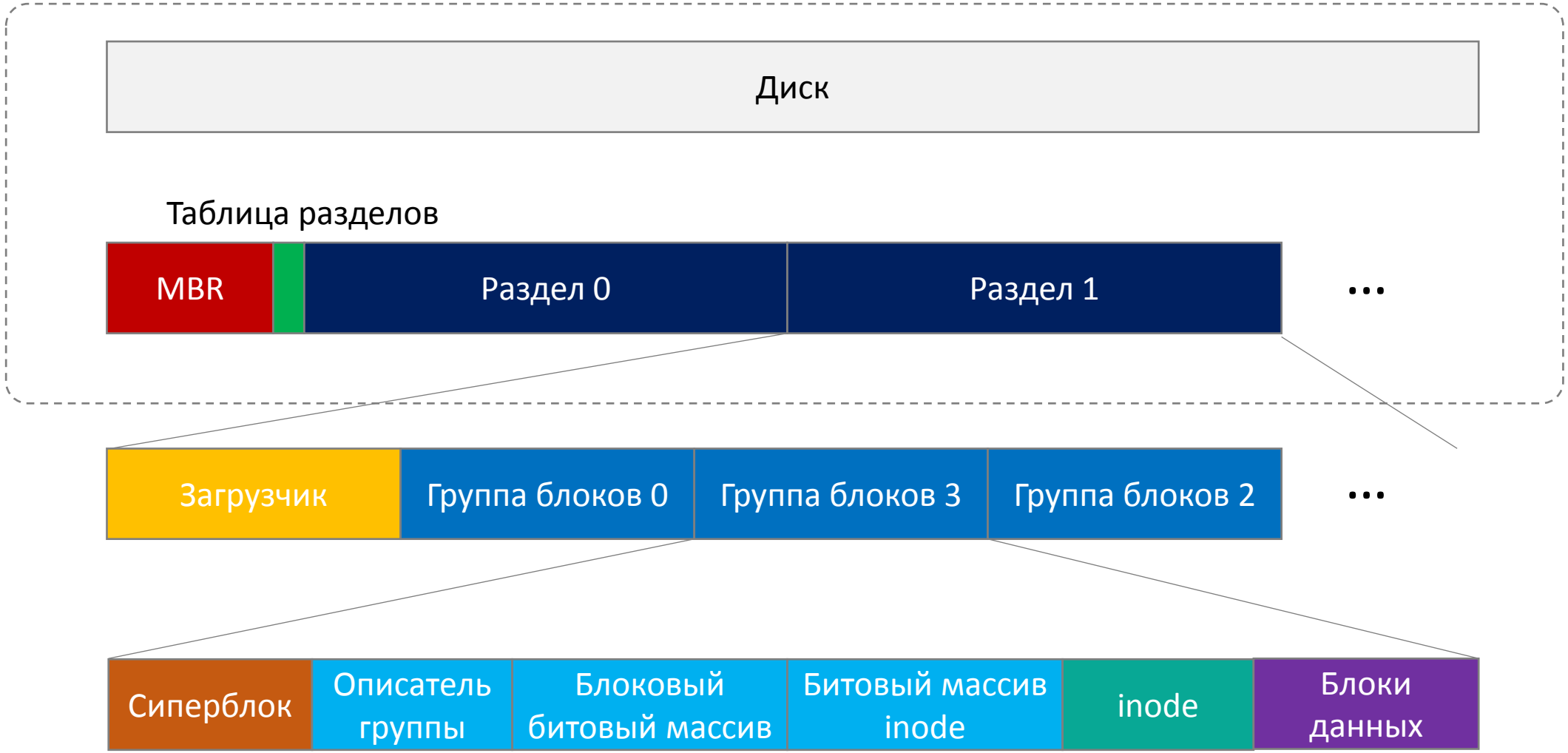
Реализованные под уровнем ВФС реальные файловые системы не обязаны использовать внутри себя точно такие же абстракции и операции. Однако они должны реализовать семантически эквивалентные операции файловой системы (такие же, как указанные для объектов ВФС). Элементы структур данных **operations** для каждого из четырех объектов ВФС — это указатели на функции в нижележащей файловой системе.

Файловые системы в Linux

Дисковое пространство (1)

- Сектор 0 на диске называется главной загрузочной записью (Master Boot Record – MBR) и используется для загрузки компьютера
- В конце MBR содержится таблица разделов. Из этой таблицы берутся начальные и конечные адреса каждого раздела
- Один из разделов в этой таблице помечается как активный. При загрузке компьютера BIOS (базовая система ввода-вывода) считывает и выполняет MBR
- Первое, что делает программа MBR, — находит расположение активного раздела, считывает его первый блок, который называется загрузочным, и выполняет его
- Программа в загрузочном блоке загружает операционную систему, содержащуюся в этом разделе

Дисковое пространство (2)



Файловая система ext2 (1)

- Первый блок — это **суперблок** (superblock), в котором хранится информация о компоновке файловой системы, включая количество inode, количество дисковых блоков, начало списка свободных дисковых блоков (это обычно несколько сотен элементов)

- Затем следует дескриптор группы, содержащий информацию о расположении битовых массивов, количестве свободных блоков и inode в группе, а также количестве каталогов в группе. Эта информация важна, так как файловая система ext2 пытается распределить каталоги равномерно по всему диску

- В двух битовых массивах ведется учет свободных блоков и свободных i-узлов. Размер каждого битового массива равен одному блоку (1КБ/4КБ). Битовые массивы используются для того, чтобы принимать быстрые решения относительно выделения места для новых данных файловой системы. Когда выделяются новые блоки файлов, то ext2 также *делает упреждающее выделение* (preallocates) нескольких (восьми) дополнительных блоков для этого же файла (чтобы минимизировать фрагментацию файла из-за будущих операций записи).

Другие файловые системы Linux:

- MINIX 1
- ext
- ext2
- ext3
- ext4

Файловая система ext2 (2)

- Затем располагаются сами inode. Они нумеруются от 1 до некоторого максимума. Размер каждого inode — 128 байт, и описывает он ровно один файл. inode содержит учетную информацию (в том числе всю возвращаемую вызовом *stat*, который просто берет ее из inode), а также достаточное количество информации для определения местоположения всех дисковых блоков, которые содержат данные файла.

- Следом за inode идут блоки данных. Здесь хранятся все файлы и каталоги. Если файл или каталог состоит более чем из одного блока, то эти блоки не обязаны быть непрерывными на диске

- Соответствующие каталогам inode разбросаны по всем группам дисковых блоков. Ext2 пытается расположить обычные файлы в той же самой группе блоков, что и родительский каталог, а файлы данных — в том же блоке, что и inode исходного файла (при условии, что там имеется достаточно места)

Файловая система ext2 (3)

Открытие файла

- вызов системной функции open
- путь к файлу разбирается, и из него извлекаются составляющие его каталоги
- если файл имеется в наличии, то система извлекает номер inode и использует его как индекс таблицы inode (на диске) для поиска соответствующего inode и считывания его в память
- этот inode помещается в таблицу inode (структуру данных ядра, которая содержит все inode для открытых в данный момент файлов и каталогов). Формат элементов inode должен содержать (как минимум) все поля, которые возвращает системный вызов stat
- создается запись в системной таблице описания открытых файлов с указанием на советующий inode
- возвращается файловый дескриптор запрашиваемого файла. Файловый дескриптор указывает на запись в таблице файловых дескрипторов процесса.

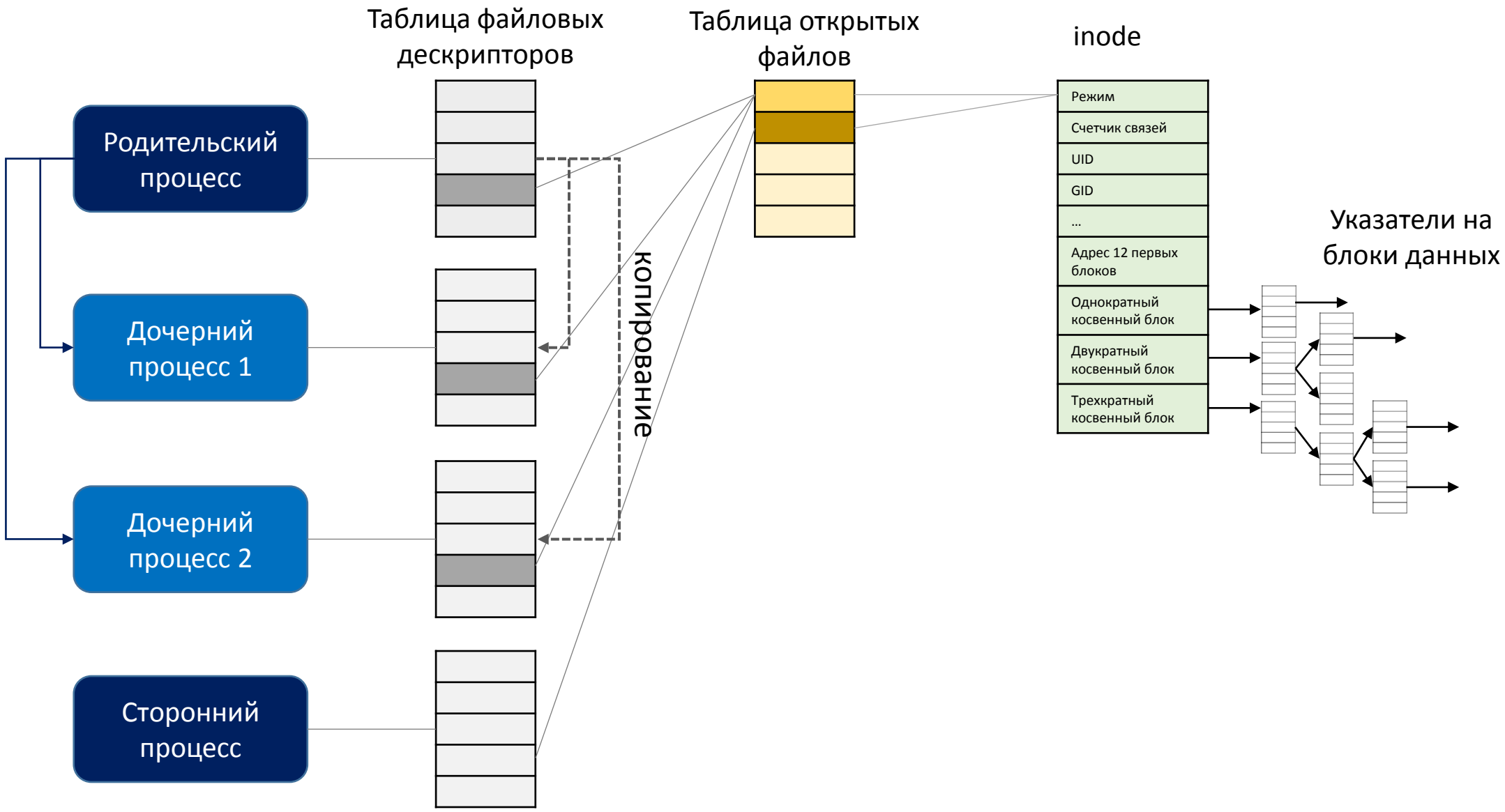
Структура inode в Linux

Поле	Размер, байт	Описание
Mode	2	Тип файла, биты защиты, биты setuid и setgid
Nlinks	2	Количество элементов каталога, указывающих на этот i-узел
Uid	2	UID владельца файла
Gid	2	GID владельца файла
Size	4	Размер файла в байтах
Addr	60	Адрес первых 12 дисковых блоков файла и 3 косвенных блоков
Gen	1	Номер «поколения» (увеличивается на единицу при каждом повторном использовании i-узла)
Atime	4	Время последнего доступа к файлу
Mtime	4	Время последней модификации файла
Ctime	4	Время последнего изменения i-узла (не считая других раз)

Чтение файла

- вызов системной функции read: `n = read(fd, buffer, nbytes)`
- ядро получает управление и обращается к таблице файловых дескрипторов (один элемент на каждый открытый файл)
- далее ядро обращается к таблице описания открытых файлов (open file description table) и ищет запись, соответствующую открытому файлу из таблицы файловых дескрипторов. Особенностью таблицы описания открытых файлов является то, что каждая запись содержит указатель, определяющий тот байт в файле, с которого начнется следующая операция чтения или записи, а также бит чтения/записи
- затем в соответствии с записью таблицы дескрипторов открытых файлов определяется inode файла
- inode содержит информацию о расположении блоков данных, которая используется для дальнейшего обращения к диску для чтения

Открытие и чтение файла



Файловая система ext4

- Основным изменением в ext4 по сравнению с ее предшественниками является использование экстентов. Экстенты представляют собой непрерывные блоки хранилища: например, 128 Мбайт непрерывных 4-килобайтовых блоков в противовес индивидуальным блокам хранения, указываемым в ext2.

- В результате ext4 может обеспечить более быстрые операции файловой системы и поддержку более объемных файлов и более крупных размеров файловой системы. Например, для размера блока в 1 Кбайт ext4 увеличивает максимальный размер файла с 16 Гбайт до 16 Тбайт

- Ext4 поддерживает журнал, который в последовательном порядке описывает все операции файловой системы. При такой последовательной записи изменений в данных файловой системы или ее метаданных (inode, суперблоке и т. д.) операции записи не страдают от издержек перемещения дисковых головок (во время случайных обращений к диску).

- Если же до фиксации изменений происходит системный сбой или отказ электропитания, то при последующем запуске система обнаружит, что файловая система не была должным образом размонтирована, просмотрит журнал и выполнит все (описанные в журнале) изменения в файловой системе.

Источники

Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.