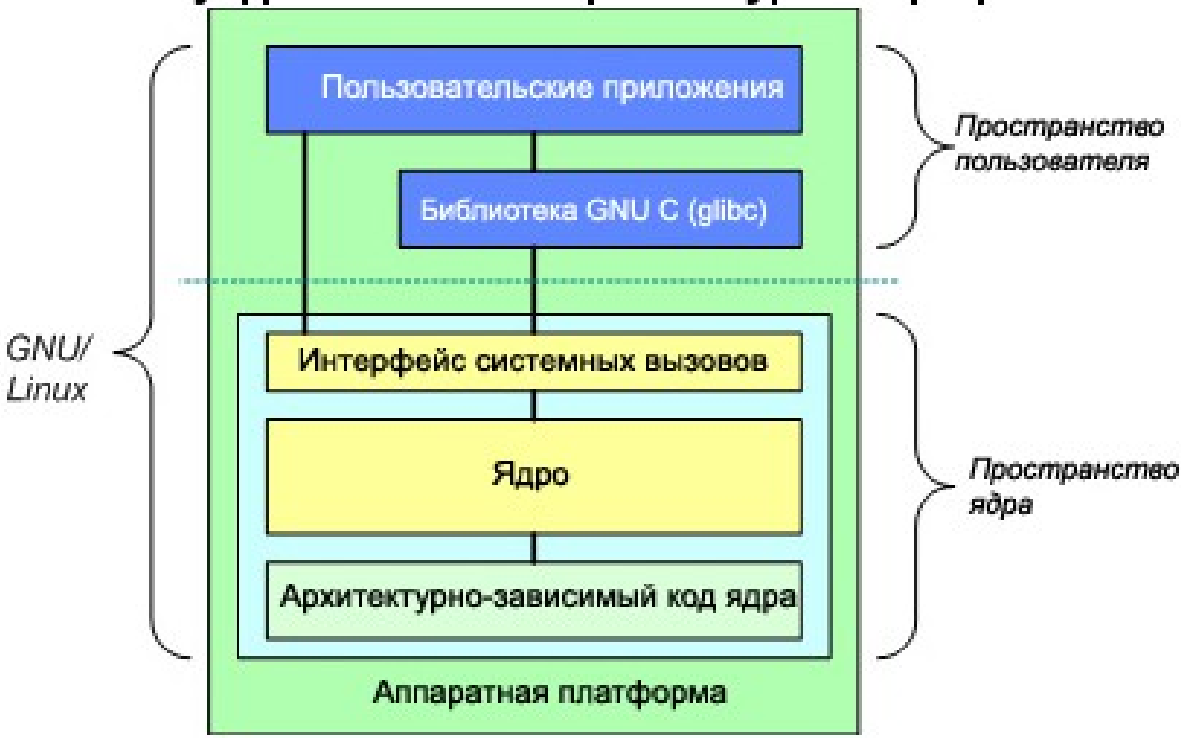
Ядро OS Linux

THE CORE

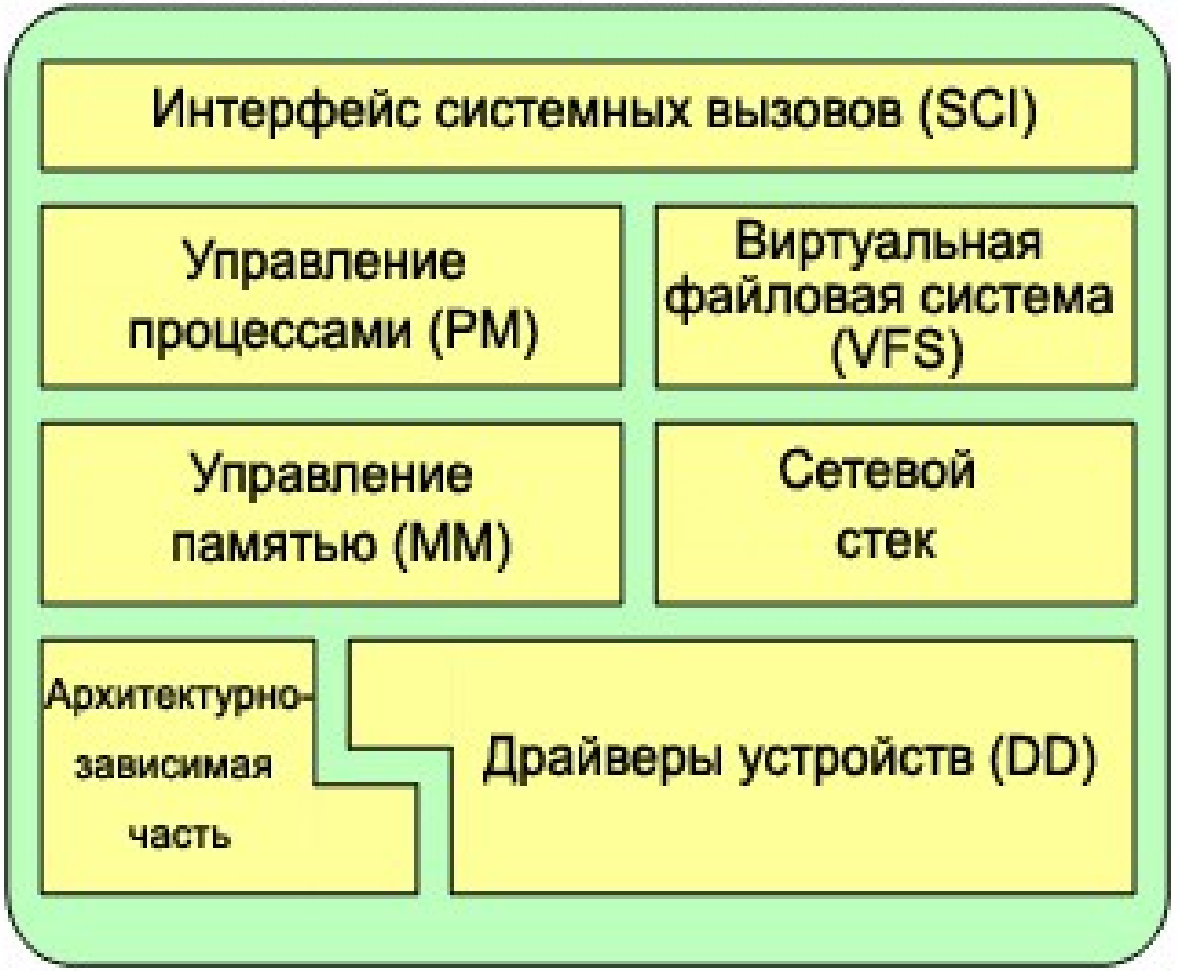
Рис. 2. Фундаментальная архитектура операционной системы GNU/Linux



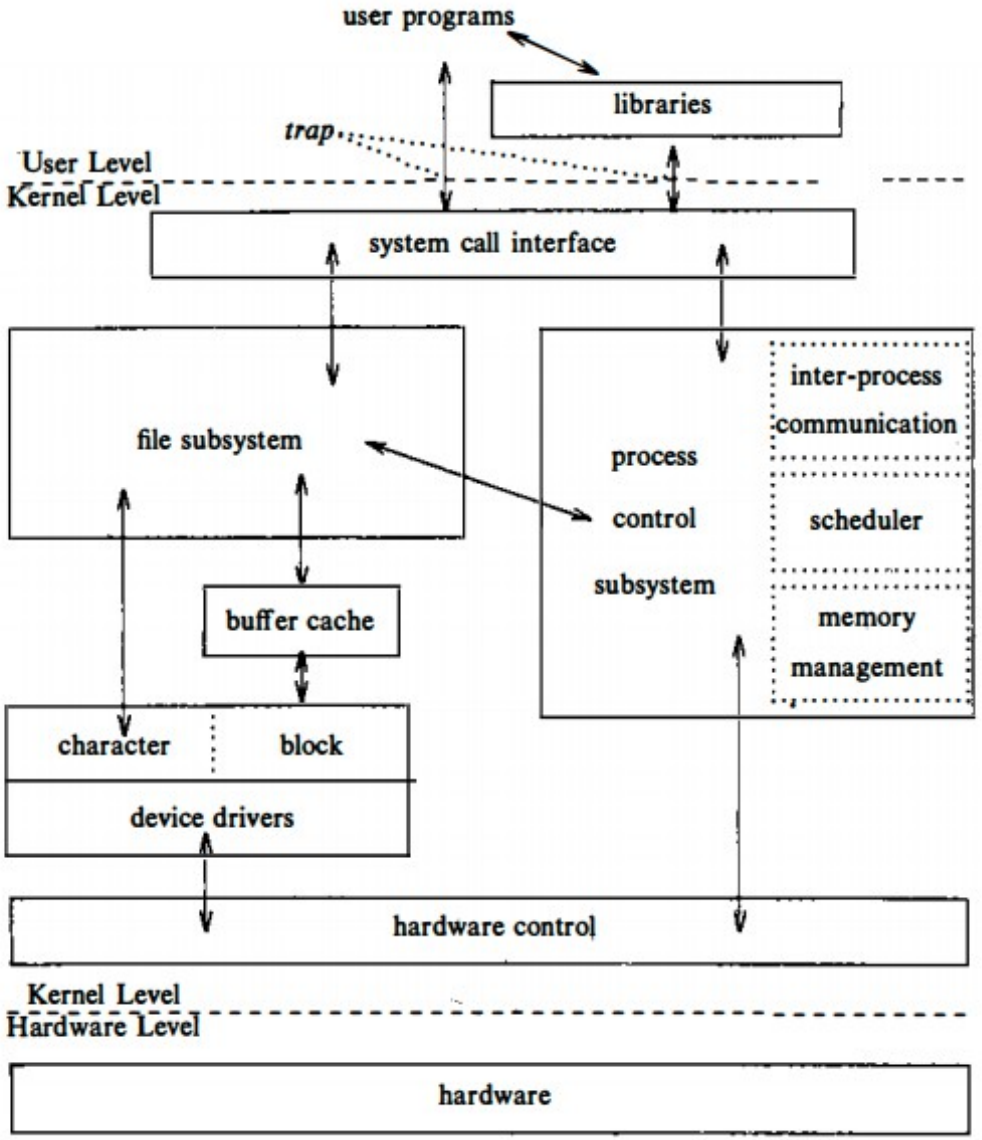
Ядро Linux можно разделить на три больших уровня. Наверху располагается **интерфейс системных вызовов**, который реализует базовые функции, например, чтение и запись. Ниже интерфейса системных вызовов располагается **архитектурно-независимый код ядра.** Этот код является общим для всех процессорных архитектур, поддерживаемых Linux. Еще ниже располагается **архитектурно-зависимый код**, образующий т.н. BSP (Board Support Package - пакет поддержки аппаратной платформы). Этот код зависит от процессора и платформы для конкретной архитектуры.

Подсистемы ядра Linux

Рис. 3. Один из возможных взглядов на архитектуру ядра Linux



Архитектура Unix OS

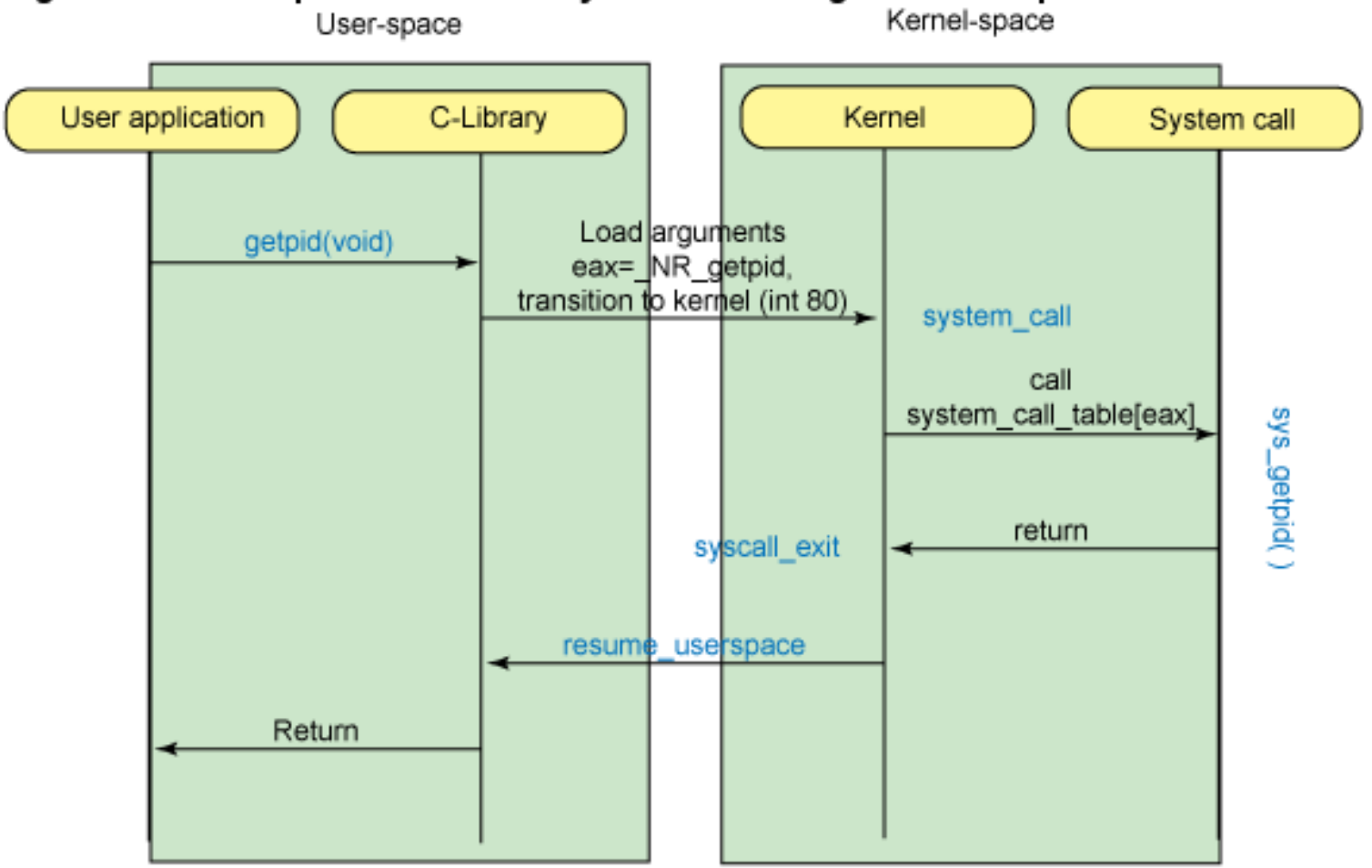


Интерфейс системных вызовов

В любой ОС есть механизм который позволяет пользовательских программам обращаться к услугам ядра.

Системные вызовы (system calls) – это интерфейс между операционной системой и пользовательской программой. Они создают, удаляют и используют различные объекты, главные из которых – процессы и файлы. Пользовательская программа запрашивает сервис у операционной системы, осуществляя системный вызов. Имеются библиотеки процедур, которые загружают машинные регистры определенными параметрами и осуществляют прерывание процессора, после чего управление передается обработчику данного вызова, входящему в ядро операционной системы. Цель таких библиотек – сделать системный вызов похожим на обычный вызов подпрограммы.

Figure 1. The simplified flow of a system call using the interrupt method



Имеется также библиотека GNU C (glibc). Она предоставляет интерфейс системных вызовов, который обеспечивает связь с ядром и дает механизм для перехода от приложения, работающего в пространстве пользователя, к ядру. Это важно, поскольку ядро и пользовательское приложение располагаются в разных защищенных адресных пространствах.

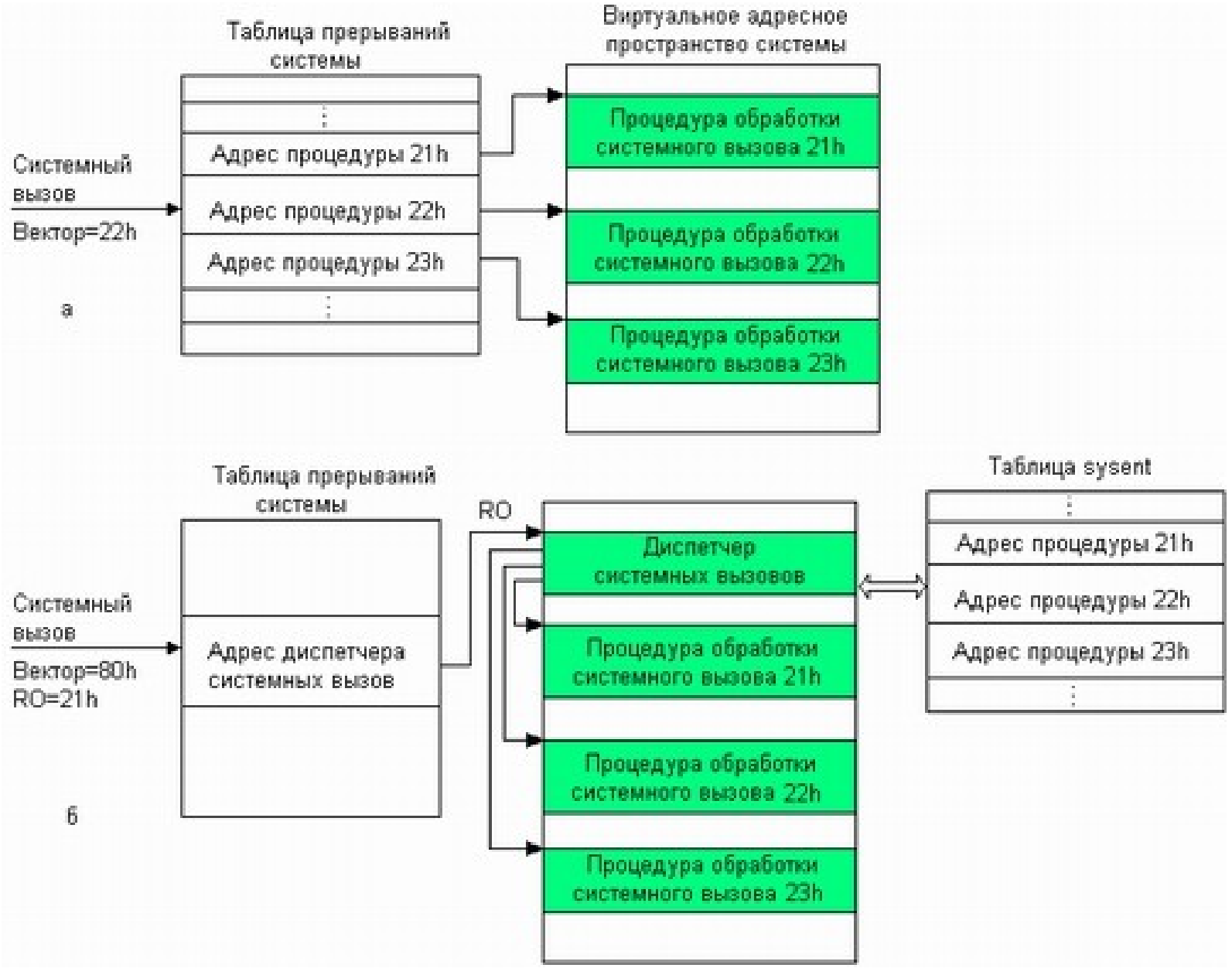


Таблица прерываний системы
Виртуальное адресное пространство системы
Системный вызов
Вектор=22h
Адрес процедуры 21h
Адрес процедуры 22h
Адрес процедуры 23h
Процедура обработки системного вызова 21h
Процедура обработки системного вызова 22h
Процедура обработки системного вызова 23h
а
Таблица прерываний системы
RO
Таблица sysent
Системный вызов
Вектор=80h
RO=21h
Адрес диспетчера системных вызов
Диспетчер системных вызовов
Адрес процедуры 21h
Адрес процедуры 22h
Адрес процедуры 23h
Процедура обработки системного вызова 21h
Процедура обработки системного вызова 22h
Процедура обработки системного вызова 23h
б

Как посмотреть какие системные вызовы использует программа?

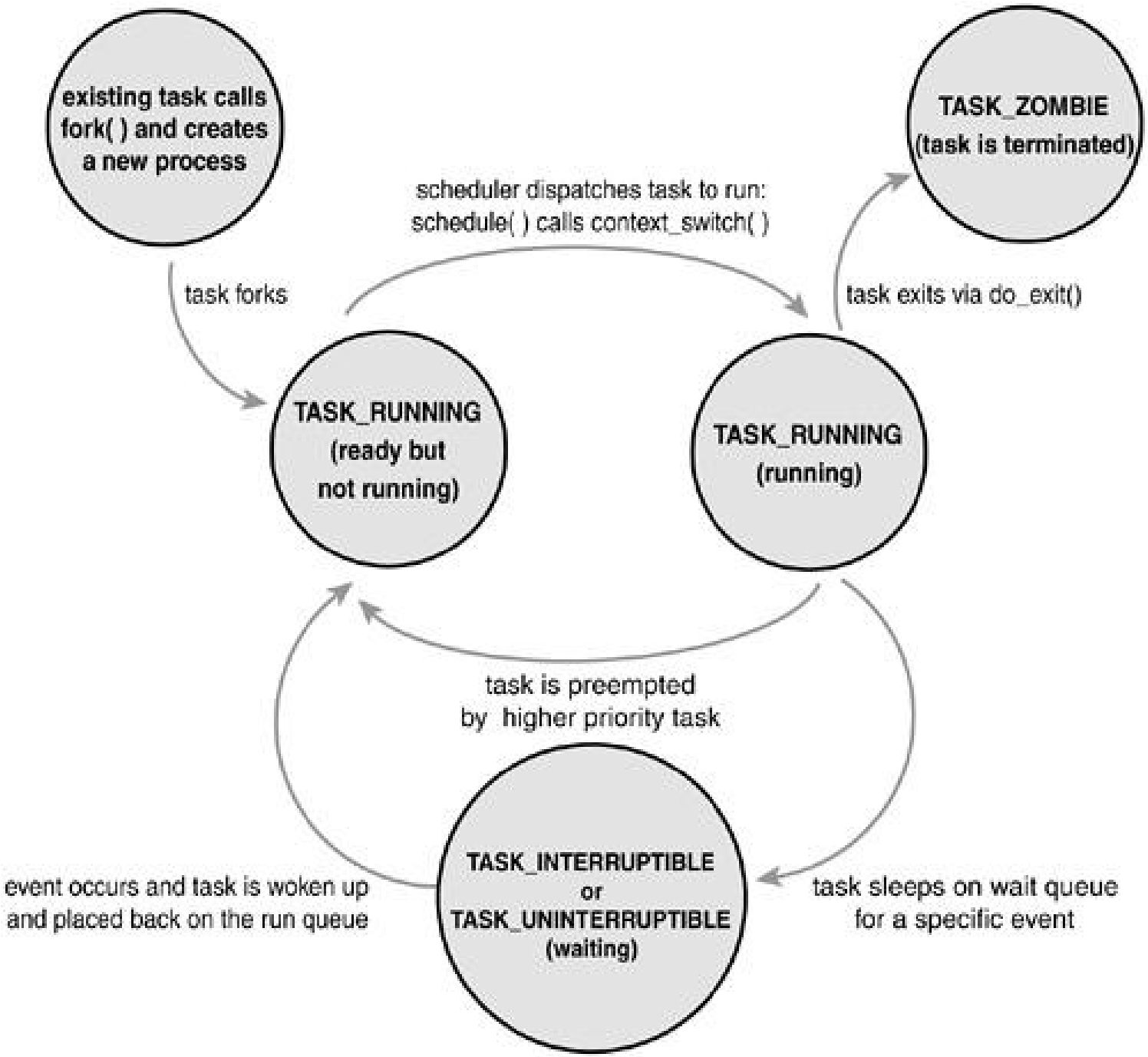
```
$>sudo apt-get install strace

$>strace ls
```

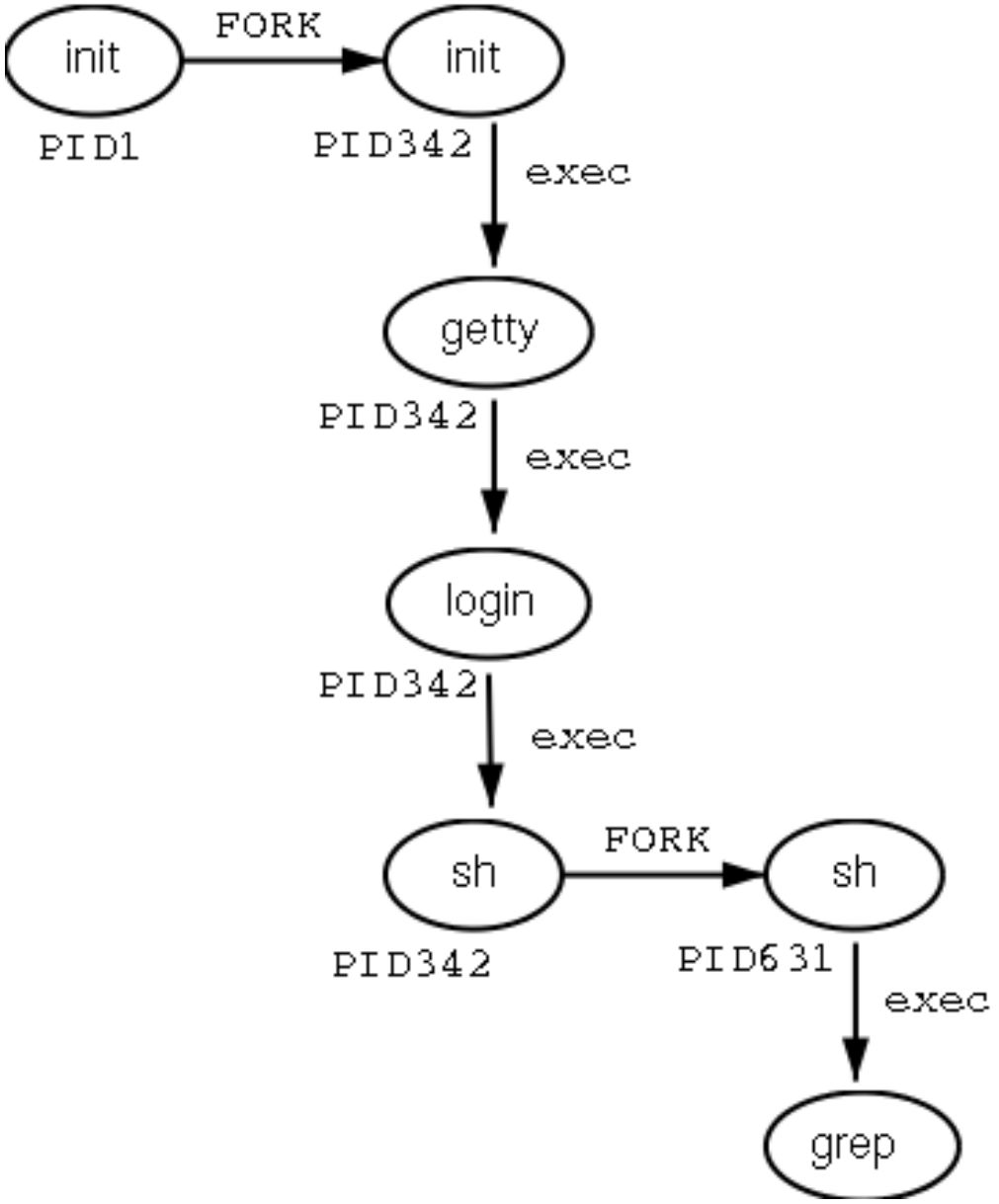
Управление процессами

Управление процессами сконцентрировано на исполнении процессов. В ядре эти процессы называются потоками (threads); они соответствуют отдельным виртуализованным объектам процессора (код потока, данные, стек, процессорные регистры). В пространстве пользователя обычно используется термин процесс, хотя в реализации Linux эти две концепции (процессы и потоки) не различают. Ядро предоставляет интерфейс программирования приложений (API) через SCI для создания нового процесса (порождения копии, запуска на исполнение, вызова функций Portable Operating System Interface [POSIX]), остановки процесса (kill, exit), взаимодействия и синхронизации между процессами (сигналы или механизмы POSIX).

Диаграмма состояний процесса


existing task calls fork( ) and creates a new process
TASK_ZOMBIE (task is terminated)
scheduler dispatches task to run: schedule( ) calls context_switch( )
task forks
task exits via do_exit()
TASK_RUNNING (ready but not running)
TASK_RUNNING (running)
task is preempted by higher priority task
TASK_INTERRUPTIBLE or TASK_UNINTERRUPTIBLE (waiting)
event occurs and task is woken up and placed back on the run queue
task sleeps on wait queue for a specific event

Системные вызовы fork(), exec()



Как посмотреть запущенные процессы в системе

```
$>ps -ef
```

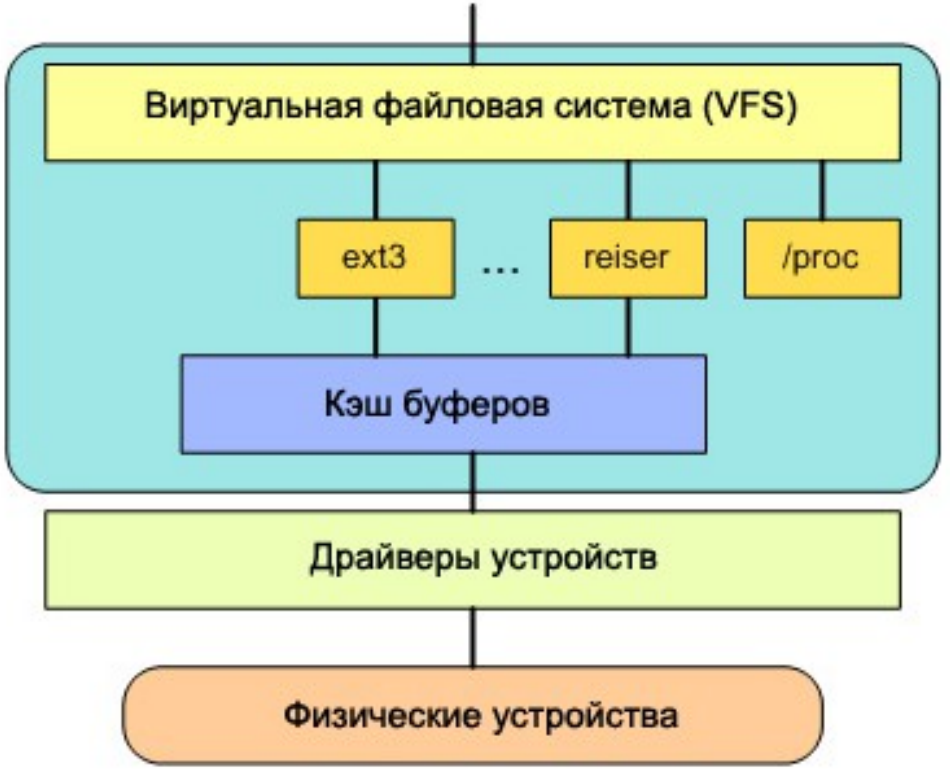
Управление памятью

Другой важный ресурс, которым управляет ядро - это память. Для повышения эффективности, учитывая механизм работы аппаратных средств с виртуальной памятью, память организуется в виде т.н. страниц (в большинстве архитектур размером 4 КБ). В Linux имеются средства для управления имеющейся памятью, а также аппаратными механизмами для установления соответствия между физической и виртуальной памятью.

Управление памятью

- Каждый процесс обладает своим виртуальным адресным пространством.
- Процессоры поддерживают страничное преобразование адресов
- Страница виртуальной памяти может соответствовать странице физической памяти, странице из файла или устройства подкачки

Рис. 4. VFS предоставляет коммутационную матрицу



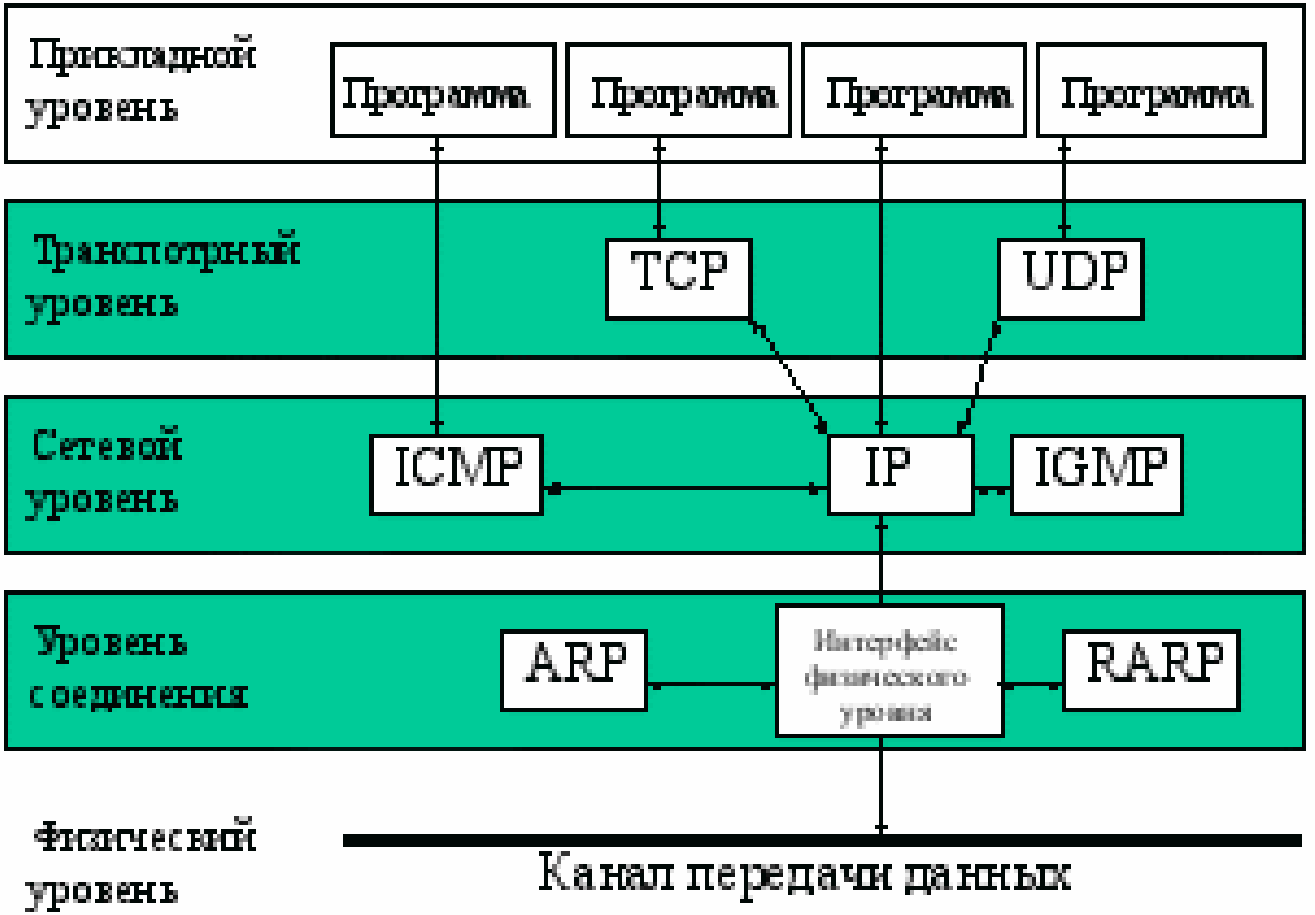
На верхнем уровне VFS располагается единая API-абстракция таких функций, как открытие, закрытие, чтение и запись файлов. На нижнем уровне VFS находятся абстракции файловых систем, которые определяют, как реализуются функции верхнего уровня. Они представляют собой подключаемые модули для конкретных файловых систем (которых существует более 50).

Ниже уровня файловой системы находится кэш буферов. Этот уровень кэширования оптимизирует доступ к физическим устройствам за счет краткосрочного хранения данных (или упреждающего чтения, обеспечивающего готовность данных к тому моменту, когда они понадобятся). Ниже кэша буферов находятся драйверы устройств, реализующие интерфейсы для конкретных физических устройств.

Сетевой стек

Сетевой стек по своей конструкции имеет многоуровневую архитектуру, повторяющую структуру самих протоколов. Вы помните, что протокол Internet Protocol (IP) - это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). Выше TCP находится уровень сокетов, вызываемый через SCI.

Стек TCP/IP



Стек протоколов TCP/IP

Драйверы устройств

Подавляющее большинство исходного кода ядра Linux приходится на драйверы устройств, обеспечивающие возможность работы с конкретными аппаратными устройствами. В дереве исходных кодов Linux имеется подкаталог драйверов, в котором, в свою очередь, имеются подкаталоги для различных типов поддерживаемых устройств, таких как Bluetooth, I2C, последовательные порты и т.д. Исходные коды драйверов устройств находятся в ./linux/drivers.

Архитектурно-зависимый код

Хотя основная часть Linux независима от архитектуры, на которой работает операционная система, в некоторых элементах для обеспечения нормальной работы и повышения эффективности необходимо учитывать архитектуру. В подкаталоге ./linux/arch находится архитектурно-зависимая часть исходного кода ядра, разделенная на ряд подкаталогов, соответствующих конкретным архитектурам. Все эти каталоги в совокупности образуют BSP. В случае обычного настольного ПК используется каталог i386. Подкаталог для каждой архитектуры содержит ряд вложенных подкаталогов, относящихся к конкретным аспектам ядра, таким как загрузка, ядро, управление памятью и т.д. Исходные коды архитектурно-зависимой части находятся в ./linux/arch.

Дальнейшее изучение

В этом обзоре мы лишь в самых общих чертах рассказали об архитектуре ядра Linux и его особенностях и возможностях. Для более глубокого изучения вам потребуется самим скомпилировать ядро.