Низкоуровневое программирование

Лекция 9

Операционные системы

Отладка

Виртуализация

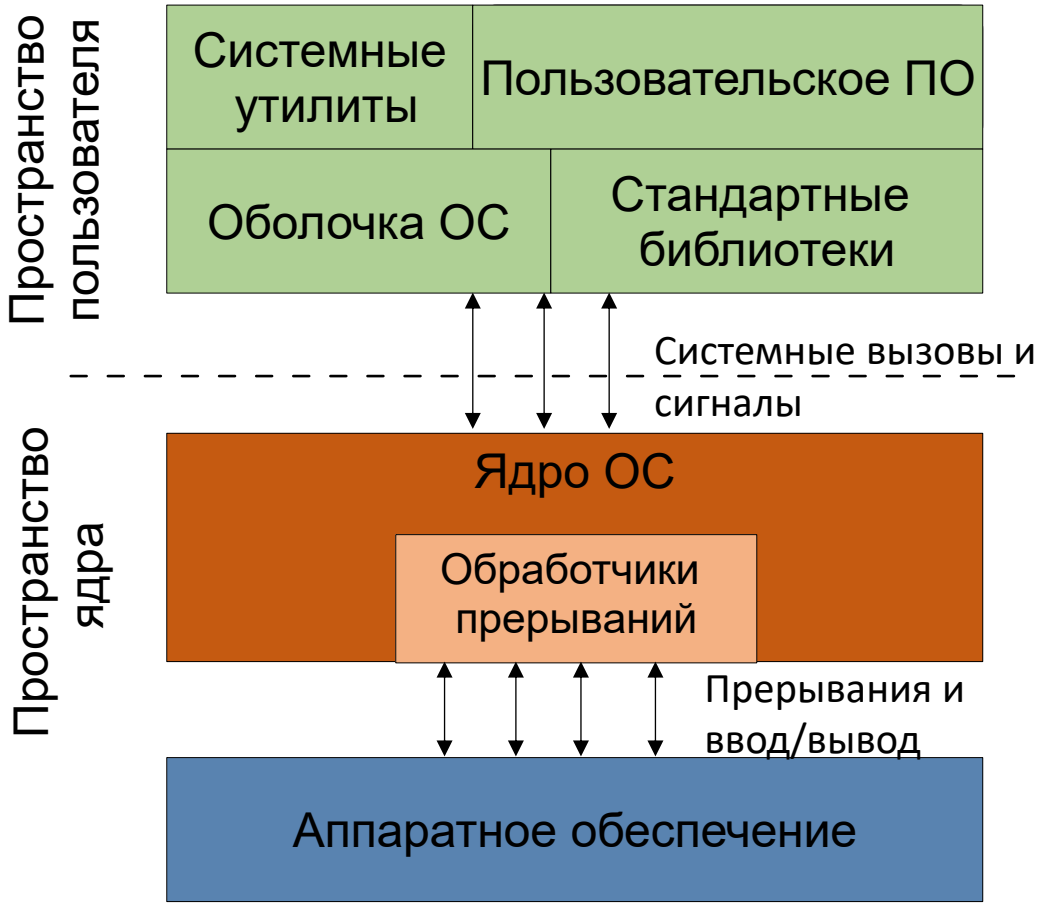
Операционные системы

Операционная система (ОС) – программный комплекс, который управляет аппаратным обеспечением компьютера и предоставляет унифицированный доступ к аппаратному обеспечению другим программам.

Ядро ОС – часть ОС, непосредственно отвечающая за управление аппаратными ресурсами компьютера и обслуживанием запросов на доступ к нему со стороны прикладных программ.

За управление конкретными устройствами отвечают специальные подключаемые модули ядра (**драйверы** в терминологии Microsoft), которые включают в себя обработчики прерываний для данных устройств.

Поскольку ядро ОС работает с оборудованием, оно должно иметь полный набор возможностей -> *его код выполняется в кольце 0*.



Пространства ядра и пользователя

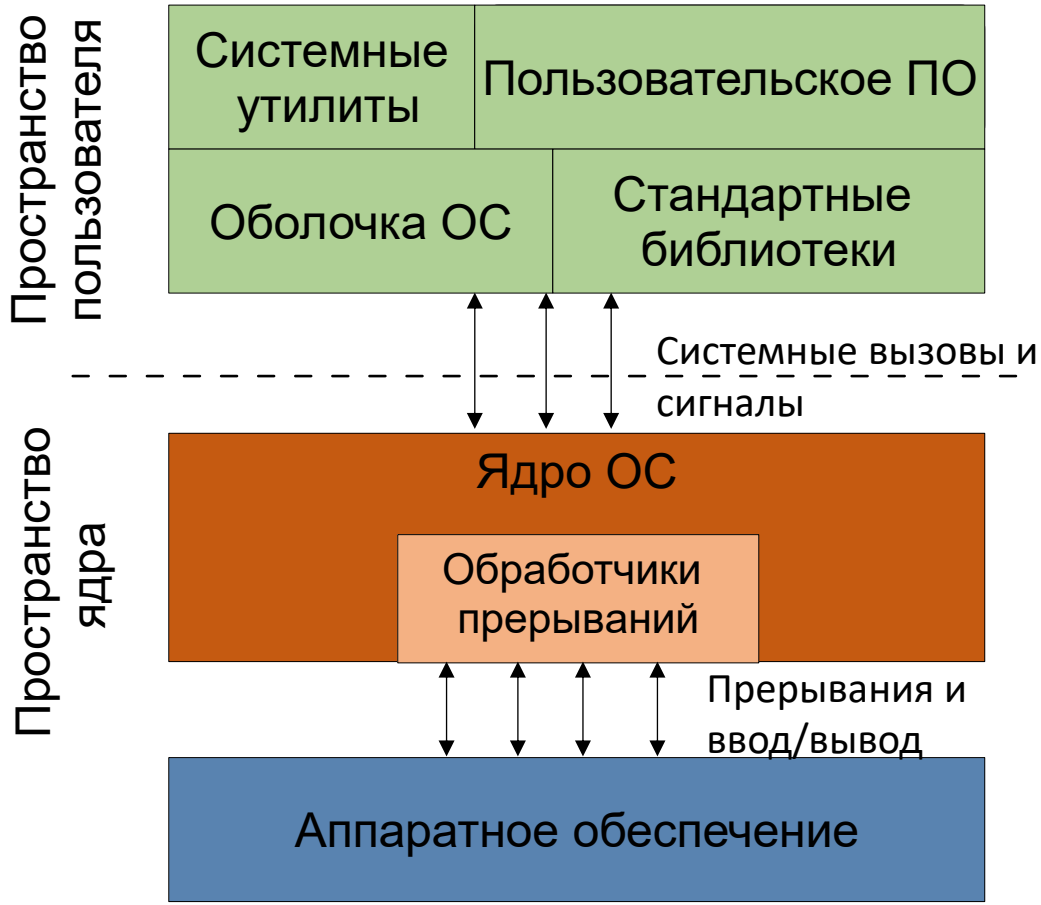
Пространством ядра называется часть адресного пространства, в которой находятся код и данные ядра ОС.

Пространством пользователя называется вся остальная часть адресного пространства.

Пространство ядра защищено средствами сегментной* и страничной** защиты – доступ к нему из пространства пользователя невозможен.

* DPL дескрипторов, связанных с ядром, равен 0 -> доступ из кольца 3 невозможен

** бит U в таблице страниц равен 0 -> доступ из кольца 3 невозможен



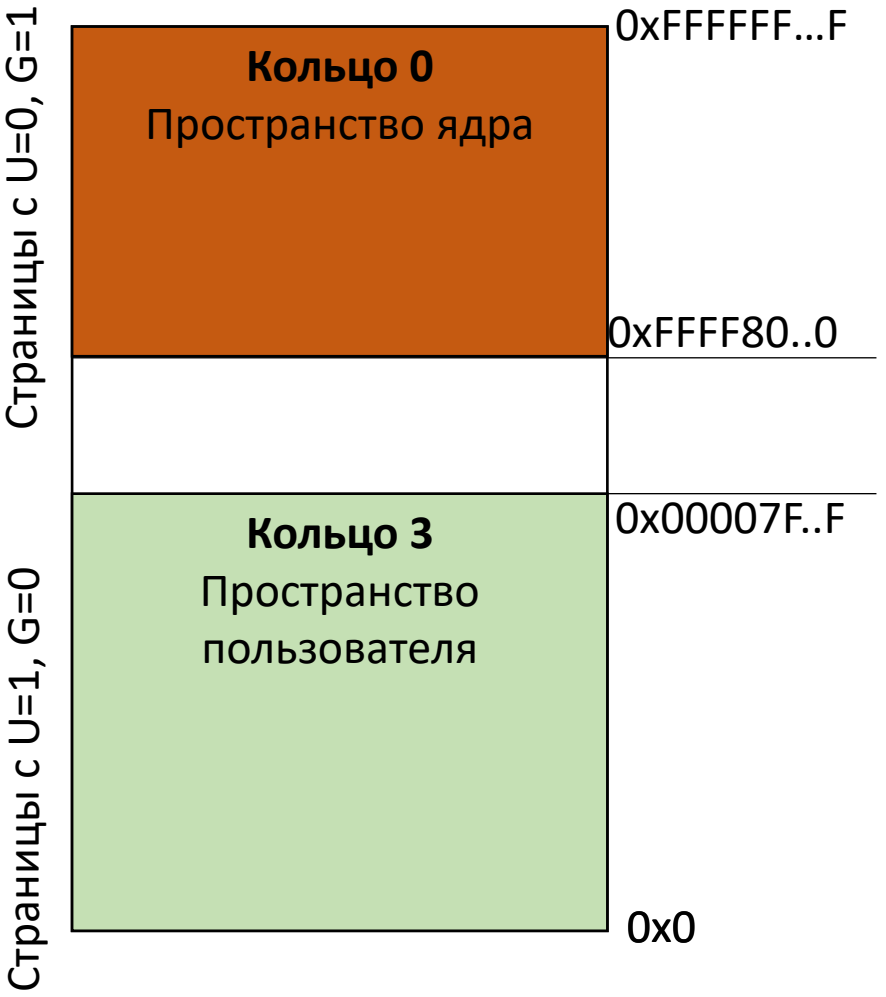
Пространства ядра и пользователя

Поскольку ядро является общим для всех программ, оно отображается в верхнюю часть адресного пространства каждого процесса. *Достигается это путем заполнения части таблицы страниц процесса одинаковыми записями* (обычно эти записи имеют флаг G=1).

При переключении между программами все адреса внутри ядра остаются валидными, что «упрощает жизнь» – иначе при каждом прерывании и/или системном вызове приходилось бы переключаться на отдельную карту страниц ядра (т.е. заменять CR3).

Переход между пространствами ядра и пользователя осуществляется посредством системных вызовов.

Набор системных вызовов определяет API (application programming interface) операционной системы.



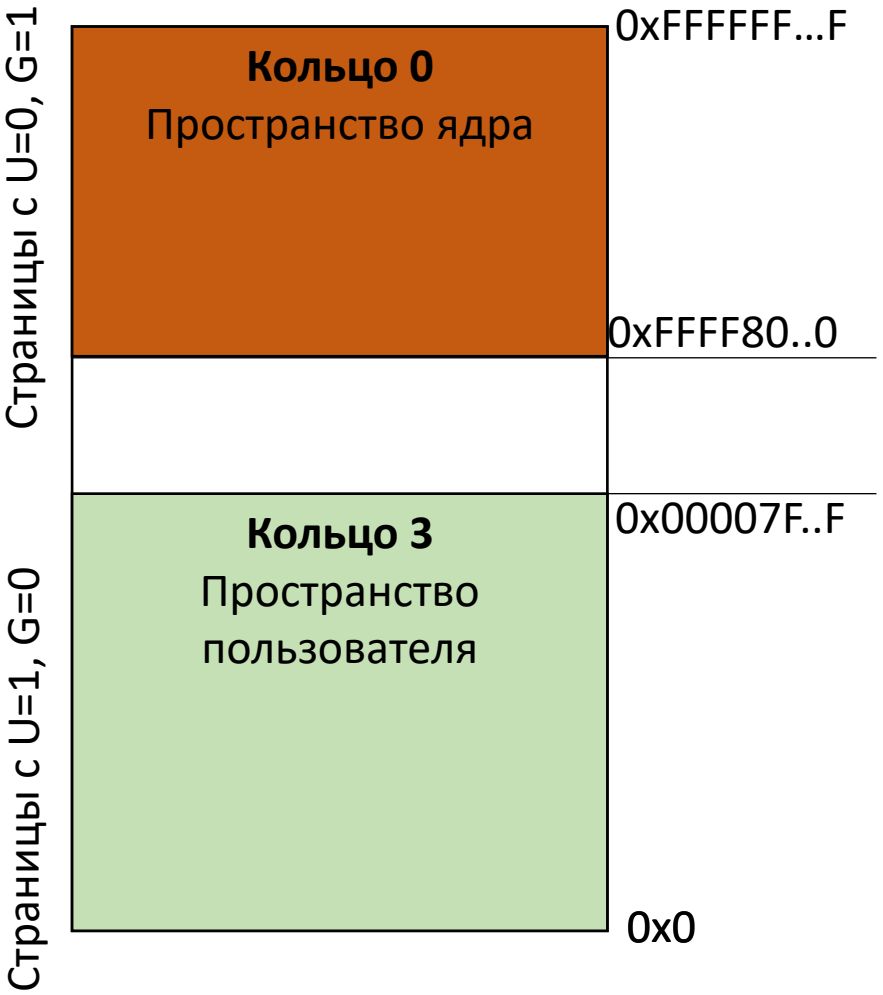
Пространства ядра и пользователя

Пространство ядра находится в области старших адресов (старшие биты адреса равны 1)

Пространство пользователя находится в младших адресах (старшие биты адреса равны 0).

В 32-битных системах граница между пространствами ядра и пользователя обычно проходит либо посередине (2 ГБ программе, 2ГБ ядру), либо по верхнему 1 ГБ (3 Гб программе).

В 64-битных системах пространства пользователя и ядра могут иметь предельный размер 2^{48} байт[1]. Между ними располагается область, которая не выделяется[2].



[1] в современных серверных ЦП 2^{57} байт, 48 и 57 обусловлены предельными размерами физической памяти

[2] адреса в данной области не являются каноническими и потому не могут использоваться

Многозадачность

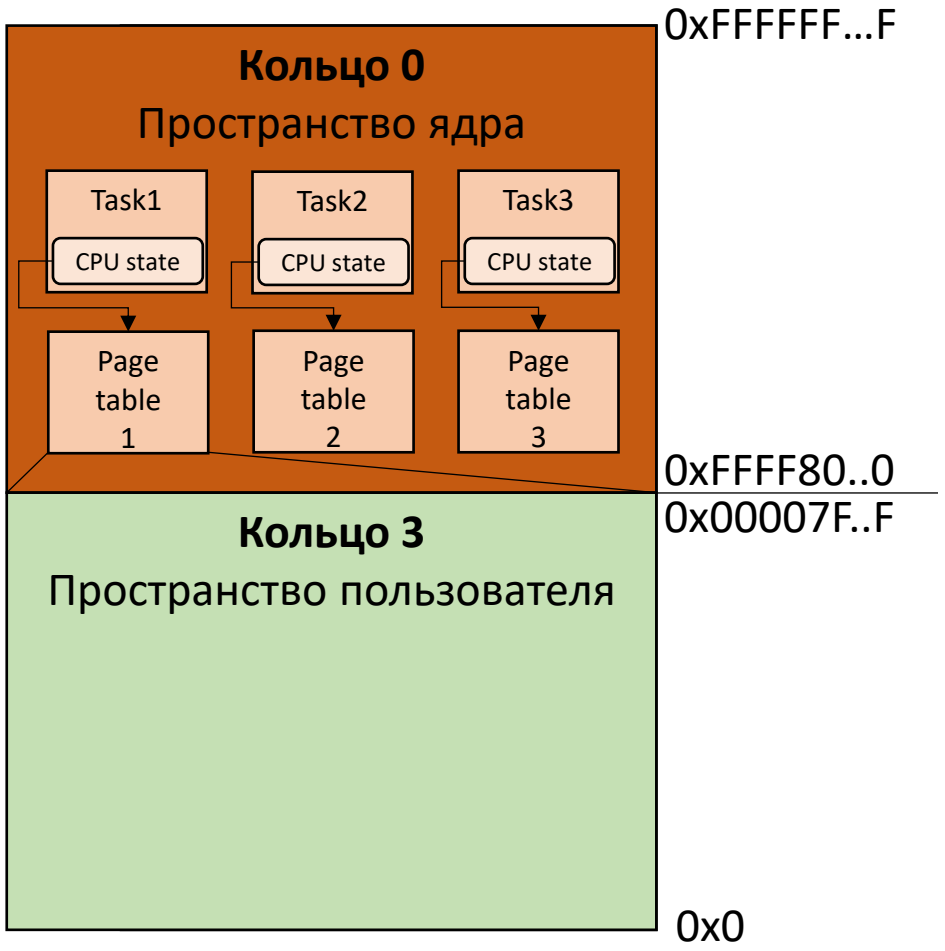
Каждой программе* в пространстве ядра соответствуют

- структура данных, которая хранит информацию о программе;
- таблица страниц.

При переключении между программами ядро ОС сохраняет состояние ЦП (т.е. значения всех регистров) в специальную область в данной структуре, выбирает программу для продолжения выполнения и загружает сохраненное состояние её структуры.

Так как при этом автоматически устанавливается значение регистра CR3, происходит переключение на другую таблицу страниц. Механизм подкачки автоматически загрузит страницы с кодом и данными программы.

**точнее, каждому потоку*



Отладчики

Отладчик – программа, предназначенная для поиска ошибок в программах.

Отладчики используют *предоставляемые ОС* средства для контроля отлаживаемой программы.

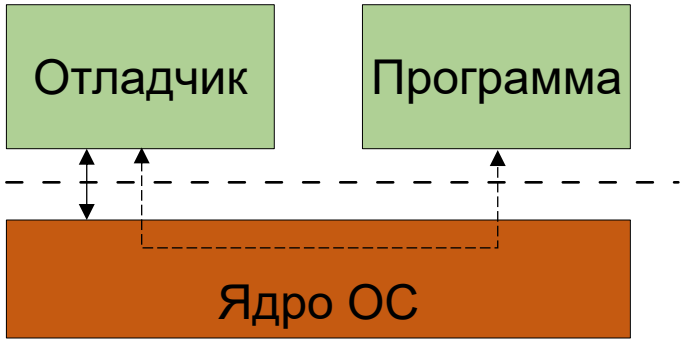
Как минимум, отладчики предоставляют функционал просмотра памяти целевой программы и контроля выполнения программы путем создания точек останова.

Точка останова (breakpoint) – место в коде программы, по достижении которого выполнение программы прерывается.

Отладчики и ОС

Обычно запущенные программы изолированы друг от друга. Однако ОС предоставляет системные вызовы и/или иные средства, которые *позволяют нарушить их изоляцию.* Обычно, есть ограничения на использование данных средств (как минимум, пользователь не должен иметь возможность вклиниться в программу другого пользователя).

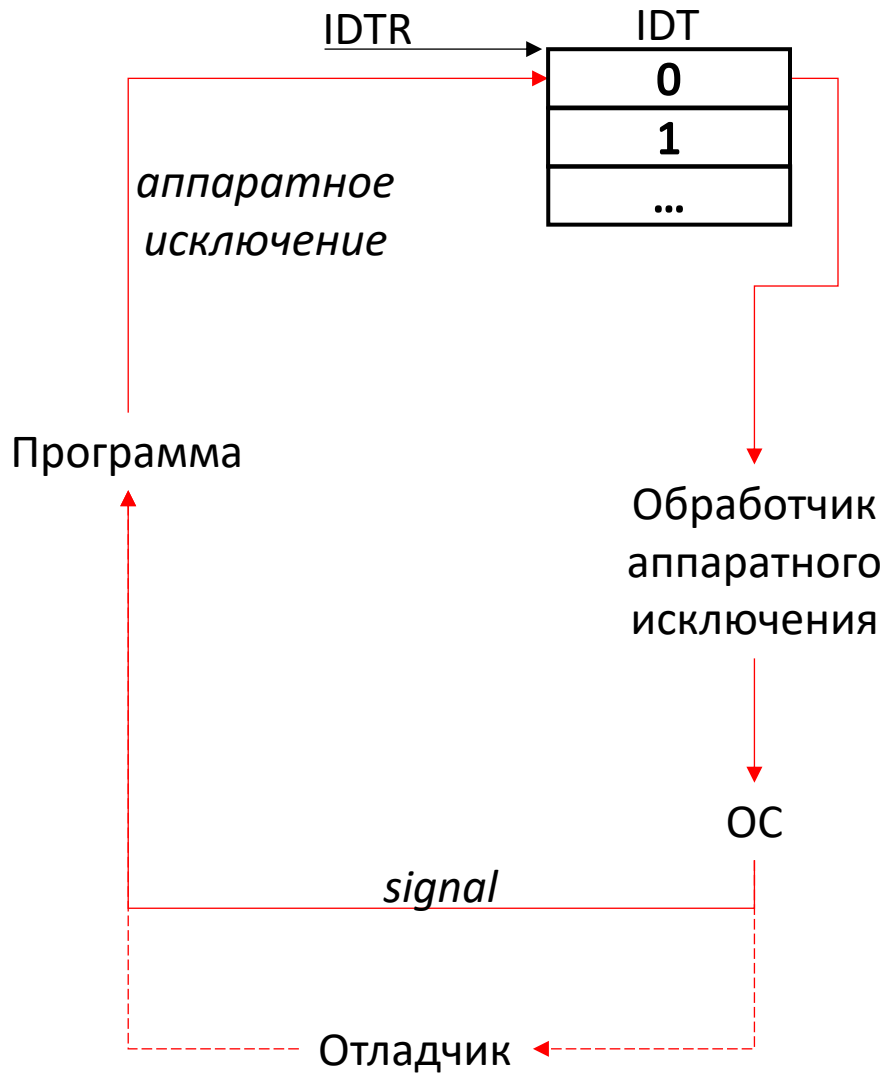
Отладчики активно эксплуатируют данные средства.



Отладчики и ОС

В случае возникновения аппаратных исключений ОС «преобразует» их в некоторый заданный формат (сигнал в UNIX, event в Windows), который отправляется программе. Если программа не готова к пришедшему сигналу, она обычно завершается с ошибкой.

Отладчики «договариваются» с ОС о том, что сигнал сначала отправляется отладчику, который принимает решение о дальнейших действиях. Программа при этом просто приостанавливается до решения отладчика.

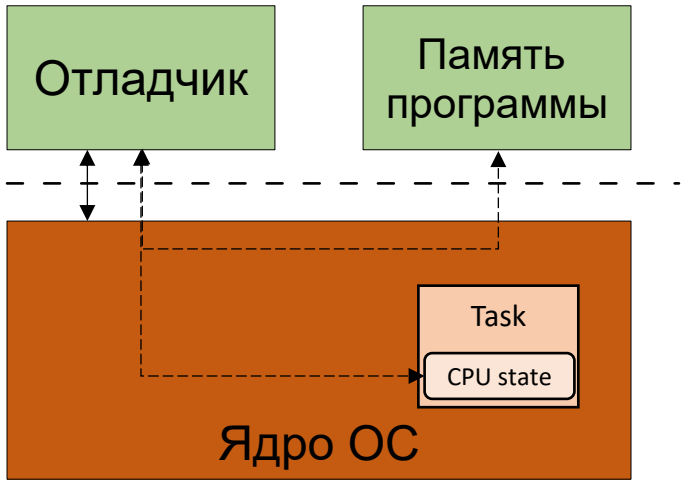


Отладчики и ОС

Отладчик также «договаривается» с ОС о доступе к состоянию приостановленной программы.

Отладчик имеет доступ:

1. К адресному пространству процесса для чтения/изменения данных и анализа исполняемого кода.
2. К сохраненному состоянию программы – для определения текущей точки выполнения (путем анализа RIP), чтения/изменения регистров общего назначения и регистра флагов (в частности, флага TF, см. далее).



Отладочные регистры

Для отладки предназначены регистры **DR0-DR7**.

Регистры DR0-3 хранят адреса точек останова.

Регистр DR6 хранит флаги состояния, выставляемые при срабатывании точки останова.

Регистр DR7 является управляющим.

Регистры DR4-5 не используются.

Доступ к отладочным регистрам возможен только из кольца 0, т.к. у данных регистров есть иные возможности, несущие угрозу безопасности.

Обычно ОС предоставляет возможность контролируемой установки данных регистров для отладки конкретной программы.

Типы точек останова

Отладочные регистры позволяют установить 3 типа точек останова:

1. На исполнение кода
2. На запись
3. На чтение или запись

Целевой адрес задается в одном из регистров DR0-DR3. Тип точки останова задается в регистре DR7.

При срабатывании точки останова генерируется аппаратное исключение 1 (#DB).

Для точек останова типа 1 исключение генерируется *до* того, как целевая инструкция будет выполнена (условие RIP==<address>).

Для точек останова 2 и 3 исключение генерируется *после* выполнения инструкции, выполнившей чтение или запись.

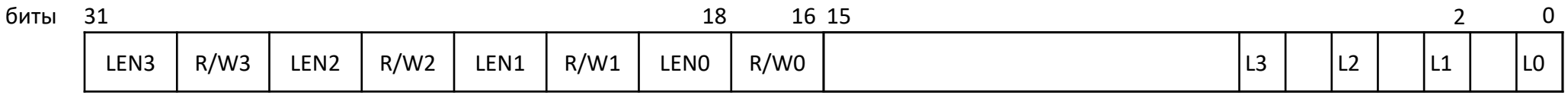
Регистр DR7

Регистр DR7 содержит ряд битовых полей, управляющих отладкой.

Флаги **L0-3** активируют точки останова по адресам в DR0-3.

Поля **R/W0-3** определяют типы точек останова в DR0-3 (00-на исполнение, 10 – на запись, 11 – на чтение и запись).

Поля **LEN0-3** определяют размер контролируемой области (00 – 1 байт, 01 – 2 байта, 10 – 8 байт, 11 – 4 байта). Начало контролируемой области совпадает с адресом в DR0-3.

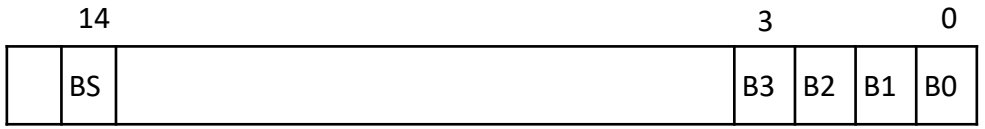


Регистр DR6

При срабатывании точки останова выставляются флаги в регистре DR6 и инициируется исключение 1 (#DB). Данный регистр затем анализируется отладчиком.

Флаги **B0-3** выставляются при срабатывании точки останова из регистра DR0-3.

Флаг **BS** выставляется, если источником #DB был флаг RFLAGS.TF=1 (см. далее).



Пошаговое выполнение

Флаг RFLAGS.TF включает режим пошагового выполнения.

При RFLAGS.TF=1 исключение #DB генерируется после выполнения *каждой инструкции*.

Обычно отладчик устанавливает этот флаг после срабатывания точки останова и снимает данный флаг, когда программист решает продолжить выполнение (Continue).

Флаг RFLAGS.TF может устанавливаться и сниматься из кольца 3.

Программная отладка

Отладочные регистры позволяют создать только 4 точки останова. Поэтому для создания обычных точек останова (на выполнение) используется иная техника.

При создании точки останова по заданному адресу отладчик:

1. читает 1 байт по заданному адресу и сохраняет его;
2. подменяет байт на значение 0xCC (инструкция INT3).

Когда исполнение доходит до точки останова, инструкция INT3 вызывает исключение 3 (#BP). Управление передается отладчику. Отладчик:

1. заменяет INT3 на исходное значение, восстанавливая исходную инструкцию;
2. вычитает 1 из значения RIP в сохраненном состоянии программы для компенсации выполненной INT3;
3. устанавливает флаг RFLAGS.TF в сохраненном состоянии программы, включая режим пошагового выполнения.

В итоге программа оказывается остановлена прямо перед выполнением инструкции, на которой находилась точка останова.

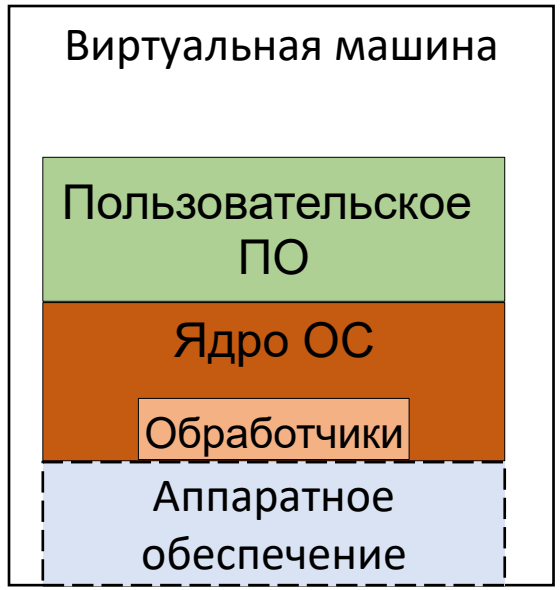
Виртуальные машины

В некоторых случаях запуск некоторой программы либо ОС целиком на реальном аппаратном обеспечении либо невозможен, либо нежелателен.

В этом случае прибегают к созданию **виртуальной машины (VM)**– программного окружения, имитирующего реальную машину.

Операционная система внутри VM называется гостевой системой (guest).

Операционная система, которая обеспечивает запуск VM, зовется хостом (host).



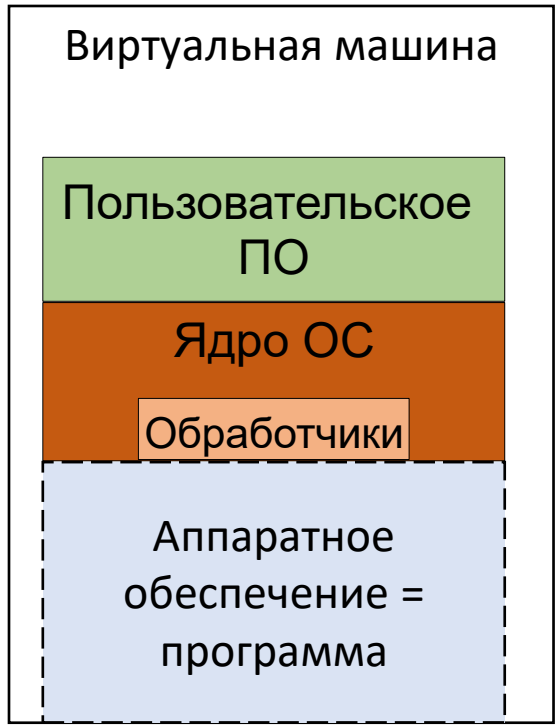
Программная эмуляция

Самым простым способом создания VM является программная эмуляция. В этом случае аппаратное обеспечение полностью эмулируется программным образом.

Эмулятор работает подобно интерпретаторам некоторых языков программирования (Python). Фактически, программа-эмулятор читает исполняемый файл и выполняет необходимые действия.

+ возможность эмулировать любое аппаратное обеспечение.

- скорость.



Аппаратная виртуализация

Если целевая архитектура процессора совпадает с архитектурой реального процессора, можно использовать аппаратную виртуализацию.

Аппаратная виртуализация предоставляет возможность отдать часть аппаратных ресурсов компьютера гостевой ОС.

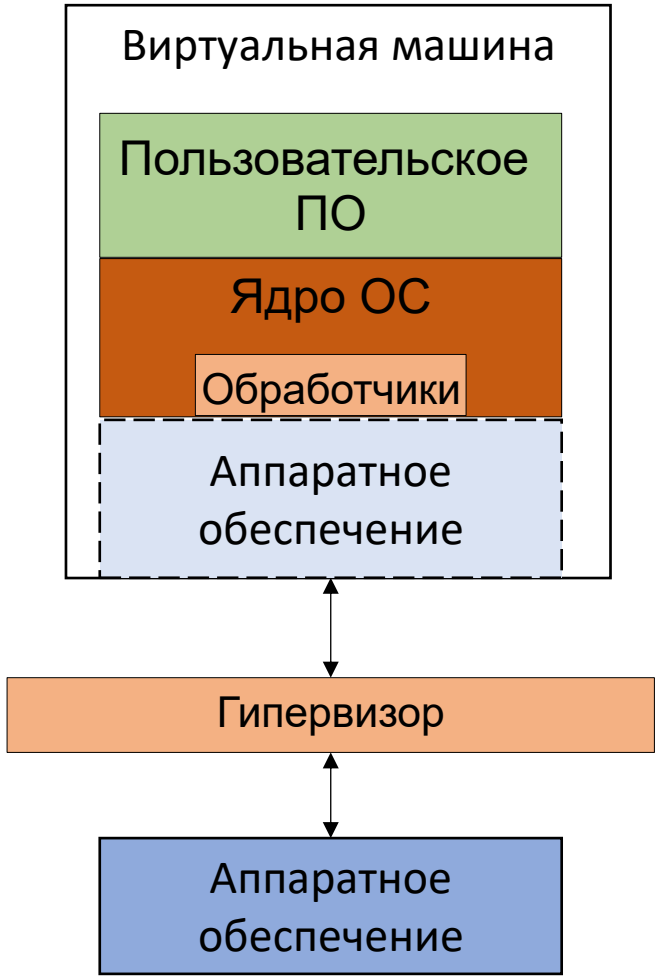
Аппаратная виртуализация использует расширения набора инструкций, специфичные для каждого производителя: VT-x и EPT в случае Intel, AMD-V и RVI в случае AMD.

Гипервизор

В случае аппаратной виртуализации за запуск и управление VM отвечает т.н. **гипервизор**.

Ядро обычной ОС занимается управлением аппаратным обеспечением и контролем обычных программ. Данный уровень полномочий иногда называют *режимом супервизора* (англ. контролер/наблюдатель).

Название гипервизора отражает тот факт, что гипервизор контролирует работу ОС (т.е. супервизора)

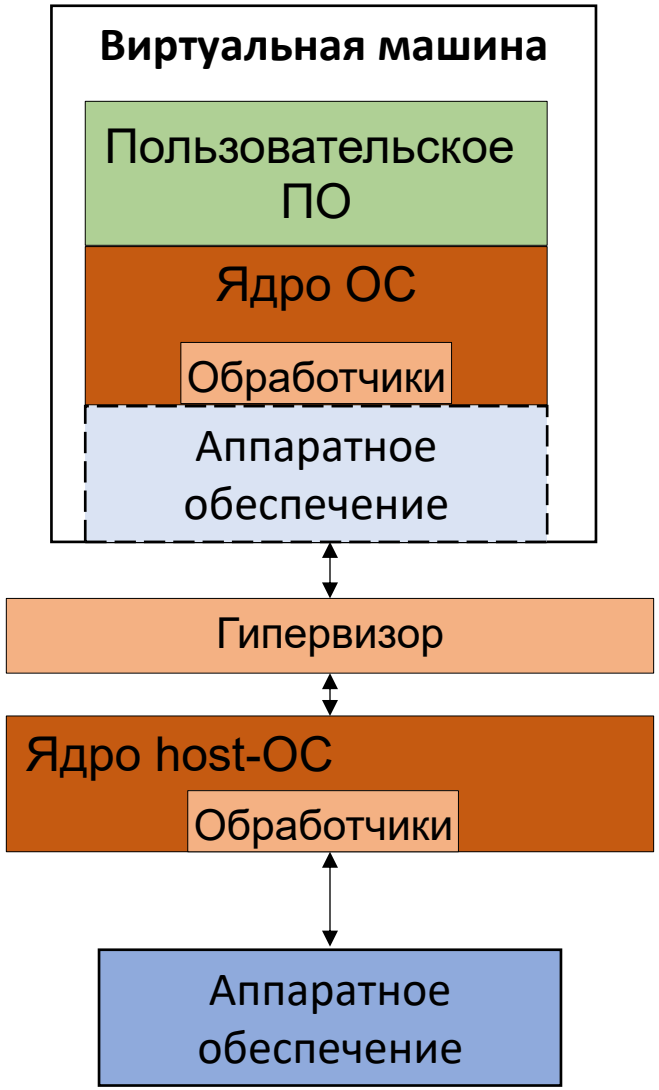


Гипервизор типа II

Наиболее распространенным вариантом является гипервизор II типа. В этом случае гипервизор является модулем ядра host-ОС или даже отдельной программой. За контроль аппаратного обеспечения и первичную обработку прерываний отвечает ядро host-ОС.

Гостевая ОС под управлением гипервизора II типа не может получить прямой доступ к аппаратному обеспечению. Гипервизор запрашивает соответствующие действия у host-ОС, а затем прозрачно передает результаты гостевой ОС.

Примеры: VMware Workstation, VirtualBox.



Гипервизор типа I

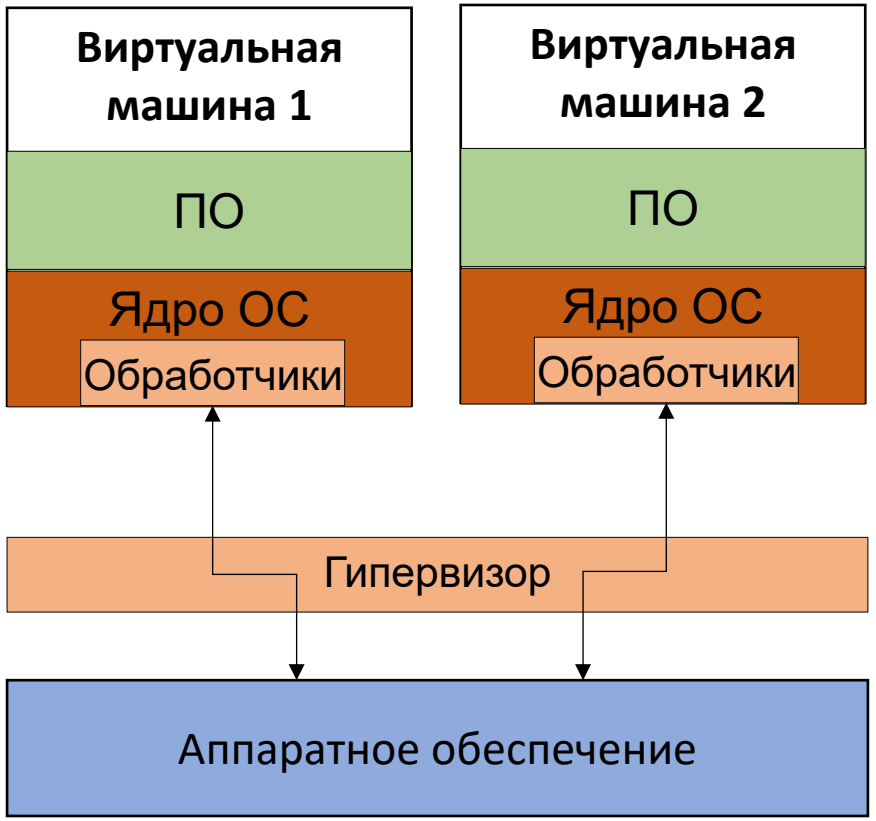
Гипервизор I типа является минимальной host-ОС, которая отвечает за распределение аппаратных ресурсом между VM.

В случае гипервизора I типа гостевая ОС обладает полным доступом к назначенному ей оборудованию (и только к нему).

Роль гипервизора сводится к контролю доступа и маршрутизации аппаратных прерываний.

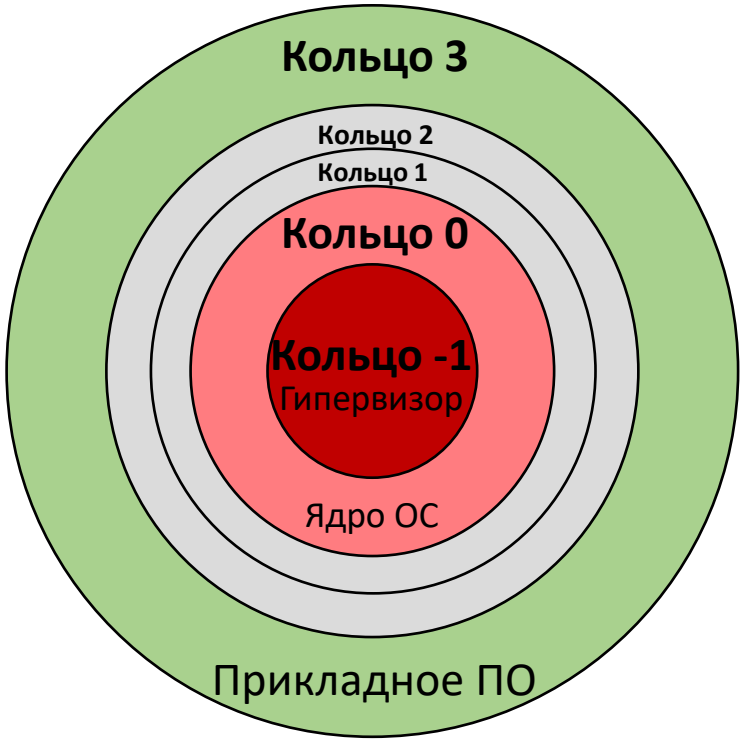
Обычно с гипервизором I типа связана одна из гостевых ОС, имеющая доступ к самому гипервизору (управляющая ОС).

Примеры: Hyper-V, Xen, VMWare ESXi.



Гипервизор типа I

Гипервизоры типа I работают в особом кольце, часто называемом **кольцо -1**. Номер кольца отображает тот факт, что гипервизор имеет наибольший уровень привилегий, превышающем уровень привилегий ядра любой из запущенных ОС (каждое из которых работает в кольце 0).



Примечание 1: еще большим уровнем привилегий обладает т.н. System Management Mode (см. Intel Management Engine и AMD Platform Security Processor), работающий в кольце -2

Примечание 2: хотя данный уровень называют кольцом -1, формально код гипервизора работает с CPL=0

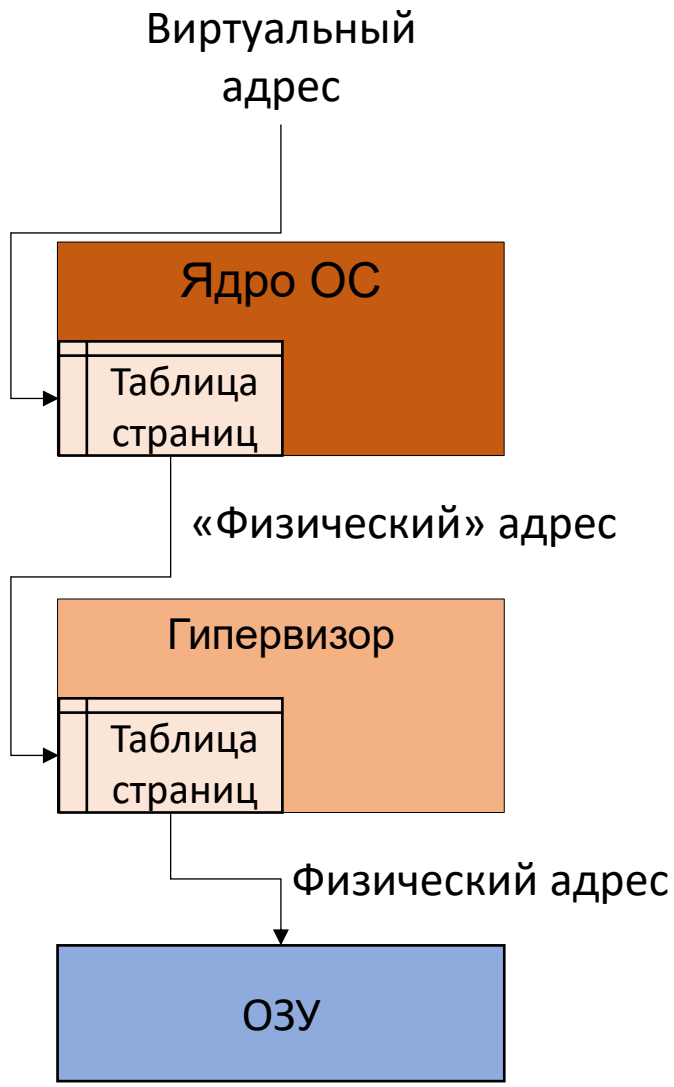
Изоляция памяти

Как и в случае с обычными программами, одна VM не должна оказывать влияния на другую VM. В первую очередь это означает, что одна VM не должна иметь доступ к памяти другой VM.

Достигается это путем ведения на уровне гипервизора дополнительных таблиц страниц, которые производят отображение
физический адрес виртуальной машины -> реальный физический адрес.

Таким образом, при использовании гипервизора, адрес транслируется дважды:
виртуальный **->** *«реальный»* **->** *реальный*.

Стоит посмотреть: shadow page tables, Intel EPT, AMD RVI



World Switch

Процесс переключения Гипервизор<->VM называется World Switch.

World Switch происходит в 3 случаях:

- при получении прерывания;
- при выполнении операций ввода-вывода;
- при выполнении некоторых привилегированных инструкций.

Список причин World Switch настраивается при запуске каждой виртуальной машины по отдельности.

Данный механизм является достаточно гибким. К примеру, гипервизор имеет возможность разрешить доступ к определенному устройству для конкретной VM без World Switch, позволив управлять устройством без ограничений. Или в обратную сторону: может запретить прямое выполнение некоторой инструкции без World Switch.

Если инструкция приводит к World Switch, то управление передается гипервизору. Гипервизор может проэмулировать выполнение операции с нужным результатом и вернуть управление VM. К примеру, таким образом можно подменить результат CPUID и заставить VM думать, что она исполняется на другом процессоре или что некоторые возможности ЦП недоступны.