Содержание

Глава 1: Общая картина

1.1 Уровни абстракции

linux можно поделить на 3 уровня обстракции: hardware → kernel → processes

1. hardware - железо, это память (RAM и ROM), это ЦП и перифирийные устройства.
2. kernel - процесс, лежащий в памяти. Посредник между пользовательскими процессами и железом.
3. processes - пользовательские процессы, которыми управляет ядро.

Kernel mode - режим работы программы ядра, позволяющее ему привелегии в виде полного доступа к УП и ОЗУ. Область памяти, доступ к которой может получить только ядро, называется **пространством ядра** (**kernel space**).

User mode - режим работы пользовательских процессов, не имеющих прямого доступа к оборудыванию. Пользовательский режим ограничен доступом к подмножеству памяти (обычно довольно небольшому) и безопасным операциям процессора. **Пользовательское пространство** - это части основной памяти, к которым могут получить доступ пользовательские процессы.

1.3 Ядро

Ядро отвечает за управление задачами в четырех основных областях системы:

- Процессы. Ядро определяет, каким процессам разрешено использовать процессор.
- Память. Ядро должно отслеживать распределение памяти: сколько в данный момент выделено конкретному процессу, сколько можно разделить между другими процессами и сколько свободно.
- Драйверы устройств. Ядро действует как интерфейс между оборудованием (например, диском) и процессами. Обычно оно управляет подключенным оборудованием.
- Системные вызовы и поддержка. Процессы обычно используют системные вызовы для связи с ядром.

1.3.1 Управление процессами

переключение контекста - это акт передачи одним процессом управления процессором другому процессу.

Ядро отвечает за переключение контекста.

как это работает

рассмотрим ситуацию, в которой процесс работает в пользовательском режиме, но его временной квант истек. Вот что происходит:

1. Процессор (фактическое оборудование) прерывает текущий процесс на основе внутреннего таймера, переключается в режим ядра и передает управление обратно ядру.
2. Ядро записывает текущее состояние процессора и памяти, что необходимо для возобновления только что прерванного процесса.
3. Ядро выполняет любые задачи, которые могли возникнуть в течение предыдущего временного кванта (например, сбор данных из ввода-вывода).
4. Теперь ядро готово к запуску другого процесса. Оно анализирует список процессов, готовых к запуску, и выбирает один из них.
5. Ядро подготавливает память для нового процесса, а затем готовит к нему процессор.
6. Ядро сообщает процессору длительность временного кванта для нового процесса.
7. Ядро переключает процессор в пользовательский режим и передает управление процессором процессу.

Переключение контекста позволяет понять, когда именно запускается ядро. Суть заключается в том, что ядро запускается между временными квантами процесса во время переключения контекста.

1.3.2 Управление памятью

Ядро должно управлять памятью во время переключения контекста, а это довольно сложная задача. Должны выполняться следующие условия:

- Ядро должно иметь в памяти выделенную область, к которой пользовательские процессы не могут получить доступ.
- Каждому пользовательскому процессу необходима собственная область памяти.
- Один пользовательский процесс не может получить доступ к области памяти, выделенной другому процессу.
- Пользовательские процессы могут совместно работать с памятью.
- Часть памяти пользовательских процессов может быть доступна только для чтения.
- Система может использовать больше памяти, чем ее существует физически, задействуя дисковое пространство в качестве вспомогательного механизма.

Современные процессоры включают в себя блок управления памятью (**memory management unit**, **MMU**), который обеспечивает схему доступа к памяти, называемую виртуальной памятью.

как это работает?

При использовании виртуальной памяти процесс не получает прямого доступа к памяти через ее физическое местоположение в компьютерной системе. Вместо этого ядро настраивает каждый процесс, чтобы он действовал так, будто ему доступна вся система. Когда процесс обращается к части своей памяти, MMU перехватывает обращение и с помощью таблицы соответствий преобразует адрес памяти с точки зрения процесса в фактическое физическое местоположение памяти в системе. Ядро по-прежнему должно инициализировать, постоянно поддерживать и изменять таблицу соответствий адресов памяти. Например, во время переключения контекста ядро должно заменить таблицу соответствий исходного процесса на таблицу последующего процесса.

Реализация таблицы соответствий адресов памяти называется **таблицей страниц**.

1.3.3 Управления драйверами устройств

Роль ядра в работе с устройствами относительно проста. Устройство обычно доступно только в режиме ядра, поскольку неправильный доступ (например, пользовательский процесс, запрашивающий отключение питания) может привести к сбою системы. Значительная проблема заключается в том, что различные устройства редко имеют один и тот же интерфейс программирования, даже если устройства выполняют одну и ту же задачу (например, две разные сетевые карты). Поэтому драйверы устройств традиционно являются частью ядра, и они стремятся представить единый интерфейс для пользовательских процессов, чтобы упростить работу разработчика программного обеспечения.

1.3.4 Системные вызовы и поддержка

Существует несколько других функций ядра, доступных пользовательским процессам. К примеру, системные вызовы (system calls, syscalls) выполняют определенные задачи, которые сам по себе пользовательский процесс выполнить не может. Например, все действия по открытию, чтению и записи файлов связаны с системными вызовами.

Ядро поддерживает пользовательские процессы с функциями, отличными от традиционных системных вызовов, наиболее распространенными из которых являются псевдоустройства. Они выглядят как устройства для пользовательских процессов, но реализуются исключительно в программном обеспечении. Это означает, что технически их не должно быть в ядре, но обычно они там присутствуют из практической необходимости. Например, устройство генератора случайных чисел ядра (/dev/random) было бы трудно безопасно реализовать с помощью пользовательского процесса.

P.S Технически пользовательский процесс, который обращается к псевдоустройству, должен задействовать системный вызов, чтобы открыть устройство, поэтому процессы не могут полностью избежать применения системных вызовов.

1.5. Пользователи

Ядро Linux поддерживает традиционную концепцию пользователя Unix. Пользователь (user) — это сущность, которая может запускать процессы и владеть файлами.

Ядро не управляет именами пользователей, оно лишь идентифицирует пользователей с помощью простых числовых идентификаторов пользователей (**user ID**, **UID**).

Пользователи существуют в основном для поддержки прав и границ в системе. Каждый процесс пользовательского пространства имеет владельца (**owner**) и, как говорят, выполняется от его имени. Пользователь может прекратить свои процессы или изменить их поведение (в определенных пределах), но не может вмешиваться в процессы других пользователей. Кроме того, пользователи могут владеть файлами и выбирать, делиться ли ими с другими пользователями.

Помимо пользователей, ассоциируемых с реальными людьми, есть и т.н. системные пользователи, например пользователи-программы или суперпользователь (**root**), имеющий полный доступ (root access) к системе. Такой пользователь традиционно считается

администратором системы.