Операционные системы hard

Автор: Вячеслав Чепелин

Содержание

1. Лекция 1

1.1. Определение операционной системы. Отличие между ОС и ядром ОС.

Операционная система (ОС) — это комплекс программ, который:

- Управляет ресурсами компьютера (процессор, память, устройства ввода-вывода).
- Предоставляет приложениям (программам в User Space) стандартный интерфейс для работы с оборудованием (Hardware).
- Обеспечивает безопасность и изоляцию между запущенными программами.

Ядро ОС (Kernel) — это центральная, главная часть операционной системы, которая работает в привилегированном режиме (Kernel Space). Именно ядро выполняет основные функции управления.

Отличие: ОС включает в себя не только ядро, но и системные утилиты, библиотеки, командные оболочки и т.д. Ядро же — это фундамент, на котором все работает.

ОС = Ядро + Системные утилиты + Пользовательские интерфейсы

1.2. Базовые понятия и концепции ОС

1.3. Общая архитектура ОС

1.4. Системные вызовы

Системный вызов — это запрос от программы в пользовательском пространстве (User Space) к ядру ОС для выполнения привилегированной операции (например, запись в файл, создание процесса, выделение памяти).

Абстрагирование: Системные вызовы скрывают от программиста сложность работы с «железом». Программа работает с абстракциями (файлы, сокеты), а ядро преобразует эти запросы в команды для конкретного оборудования.

Стабильность: Интерфейс системных вызовов очень стабилен и редко меняется (для обратной совместимости). Новые вызовы добавляются, но старые остаются. Если старые и удаляются, то сначала они остаются deprecated, а потом удаляются

1.5. Особенности параметризации системных вызовов

1.6. Режимы (modes) исполнения в ОС

Режимы работы CPU (уровни привилегий):

- Режим гипервизора (Hypervisor Mode): для запуска виртуальных машин.
- Режим ядра (Kernel Mode): код с высокими привилегиями (полный контроль над CPU, доступ ко всей памяти).
- Пользовательский режим (User Mode): ограниченные привилегии для приложений.

Попытка выполнения привилегированной инструкции (например, отключение прерываний) в пользовательском режиме вызывает исключение (exception), которое обрабатывается ядром.

1.7. Пространства памяти в ОС

- Пространство ядра (Kernel Space): защищено от прямого доступа пользовательских приложений.
- Пользовательское пространство (User Space): доступно для приложений; код ядра может обращаться к нему напрямую.

Используется виртуальная память (virtual memory) с механизмом страничной адресации (paging).

Ядерные уязвимости или exploit -

Спекулятивное исполнение - процессор исполняет обе ветки вне зависимости

1.8. Монолитное и микро-ядро ОС – различия

	Монолитное ядро (Monolithic)	**Микроядро (Microkernel)**
Структура	Все подсистемы (модули системного ядра) работают в одном адресном пространстве (ядре).	Минимальное ядро (IPC, планирование); основные сервисы работают в пользовательском пространстве
Производительность	Выше (низкие накладные расходы на взаимодействие)	Ниже из-за затрат на передачу сообщений между процессами.
Надежность	Сбой в драйвере может «уронить» всю систему	Выше: сбой службы в пользовательском пространстве не крашит ядро.
Примеры	Примеры Linux, старые версии Windows	QNX, Minix

TODO вставить рисунки с презентации

1.9. Модульная структура ядра ОС

Монолитные ядра стали модульными (Linux):

- Подсистемы разделены логически.
- Поддержка загружаемых модулей ядра (Loadable Kernel Modules). Удобно, если начинает ломаться компонента - перезапустим
- Четкие API между компонентами.

Гибридные ядра (Hybrid) – маркетинговый термин (например, Windows), где многие сервисы работают в режиме ядра, как в монолитной архитектуре.

1.10. Реализации мульти-обработки в ОС

Многопроцессорность (Multi-processing): ОС поддерживает параллельное выполнение множества процессов.

Для этого нужны ядра и потоки TODO: более подробно

планировщик и другие процессы

1.11. Различие между кооперативным и вытесняющей (preemptive) мульти-обработкой

- Кооперативная (Cooperative): Процесс добровольно возвращает управление ОС (риск: «зависший» процесс блокирует систему). В том числе Starwation - захват всех ресуросв
- Вытесняющая (Preemptive): Ядро принудительно забирает управление у процесса после исчерпания кванта времени (time slice).

1.12. Кооперативные ядра (на примере Линукса) или вытесняющие ядра (на примере Линукса)

Вытесняемость самого ядра (Kernel Preemption) – это отдельное понятие:

- Невытесняемое ядро (Non-preemptive): Процесс, выполняющий системный вызов (в режиме ядра), не может быть прерван, даже если появился более приоритетный процесс. (Пример: Linux 2.4).
- Вытесняемое ядро (Preemptive): Процесс может быть прерван даже в режиме ядра. Это необходимо для систем реального времени (Real-Time) и улучшает отзывчивость. (Пример: Linux 2.6+).

Todo: более подробно про кооперативные ядра

1.13. Ассиметричная мульти-обработка (Asymmetric multi-processing)

Asymmetric Multi-Processing (AMP): на одном процессоре ядро, на другом остальные

1.14. Симетричная мульти-обработка (symmetric multi-processing)

Symmetric Multi-Processing (SMP): Все процессорные ядра равноправны и имеют доступ к общей памяти. Это стандартная архитектура для современных ОС, включая Lin

1.15. Масштабировяния ядра ОС по процессорам

Масштабируемость ядра: Производительность ядра должна расти с увеличением числа ядер.

Методы: lock-free алгоритмы, мелкогранулярные блокировки (fine-grained locking).

1.16. Адресные пространства памяти – физическое и виртуальное

Адресные пространства:

- Физическое адресное пространство: RAM и память периферийных устройств.
- Виртуальное адресное пространство: «Вид» памяти с точки зрения процесса. Ядро управляет отображением виртуальных страниц на физические.

1.17. Таблицы трансляции виртуальных адресов

TODO, это вроде мы и так знаеми

TODO написать про страницы, про хранение таблицы, как дерево выглядит и тп

1.18. Контексты исполнения

Контекст процесса (Process Context):

- Код, выполняемый в результате системного вызова. Имеет доступ к памяти процесса.

Контекст прерывания (Interrupt Context):

- Код обработчика прерывания (Interrupt Handler). Выполняется асинхронно, не привязан к конкретному процессу, имеет серьезные ограничения (не может «засыпать»).

1.19. Стеки пользовательского кода, кода ядра и кода прерываний

Todo: для чего надо стек

- Пользовательский стек: Используется процессом в user mode.
- Стек ядра (Kernel Stack): Каждый процесс имеет свой небольшой стек ядра (например, 8 КБ) для выполнения системных вызовов. Размер фиксирован при компиляции + сконфигурирован во время -> важно избегать глубокой рекурсии и больших allocations на стеке.
- Стек прерываний (Interrupt Stack): Отдельные стеки для обработки прерываний на каждом CPU.

1.20. Страничная организация памяти и вытеснение страниц на диск

Todo

2. Лекция 2

2.1. Общая структура кода ядра Линукса

- Язык: Преимущественно C, с минимальными вставками ассемблера для критичных по производительности или низкоуровневых архитектурно-зависимых операций.
- В основном драйвера много места занимают.

Основные директории:

- arch/ (Архитектурно-зависимый код): Содержит поддиректории для каждой поддерживаемой архитектуры (x86, ARM, AARCH64 и т.д.). Включает код инициализации, обработку прерываний, низкоуровневые функции (например, memcpy), управление виртуальной памятью.
- kernel/ (Ядро ОС): Основные подсистемы: планировщик (sched/), управление процессами, системные вызовы, таймеры, синхронизация (locking/).
- mm/ (Управление памятью): Реализация виртуальной памяти, аллокаторы, подкачка.
- drivers/ (Драйверы устройств): Все драйверы, организованные по типу устройств (сеть, USB, ввод/вывод и т.д.).
- fs/ (Файловые системы): Виртуальная файловая система (VFS) и реализации конкретных ФС (ext4, NTFS, procfs).
- net/ (Сетевой стек): Реализация сетевых протоколов (IPv4/IPv6, TCP, UDP, Ethernet).
- ipc/ (Межпроцессное взаимодействие): Реализация механизмов IPC (System V, POSIX).
- include/ (Заголовочные файлы).

2.2. Управление процессами в ОС

Процесс — это программа в момент выполнения. Абстракция, инкапсулирующая ресурсы.

Ресурсы процесса:

- Адресное пространство.
- Открытые файлы и файловые дескрипторы.
- Точка выполнения (одна или несколько).
- Сигналы, права доступа, сокеты и т.д.

Реализация в Linux: Процессы и потоки представлены структурой struct task_struct (дескриптор задачи/процесса). Эта структура содержит всю информацию о процессе: состояние, идентификатор (PID), указатели на ресурсы (память, файлы), приоритет планирования и т.д.

todo: отличие потока от процесса

2.3. Управление памятью в ОС

Физическая память: Ядро управляет оперативной памятью через «Buddy Allocator», который борется с фрагментацией, выделяя память блоками кратных степеням двойки.

Виртуальная память: Каждый процесс работает в своем изолированном виртуальном адресном пространстве.

Страничная организация (Paging): Виртуальная память делится на страницы, которые отображаются на фреймы физической памяти.

Подкачка (Swapping): «Холодные» страницы могут быть вытеснены на диск (в swap-раздел), чтобы освободить оперативную память.

Отложенная загрузка (Demand Paging): Страницы загружаются в память только при первом обращении к ним.

Копирование при записи (Copy-on-Write, CoW): Механизм, используемый при fork(), когда страницы памяти между родителем и потомком разделяются до тех пор, пока один из процессов не попытается изменить их, что приводит к созданию физической копии.

Аллокаторы памяти:

- Для ядра: SLAB / SLUB аллокаторы (надстройки над Buddy Allocator) для эффективного выделения мелких объектов.
- vmalloc(): Выделяет виртуально непрерывную, но физически разрозненную память. Подходит для больших объемов, где не критична производительность TLB.
- Для пользователя: Системные вызовы brk(), mmap().

2.4. Виртуальная файловая система и управление блочным вводом-выводом

TODO: более подробно и не занудно расписать

VFS — это слой абстракции между ядром и конкретными файловыми системами (ext4, Btrfs, FAT). Она предоставляет унифицированный API для операций с файлами (open, read, write, close).

Основные структуры VFS:

- superblock: Представляет смонтированную файловую систему.
- inode (index node): Представляет объект в ФС (файл, директорию, симлинк). Содержит метаданные и указатели на блоки данных.
- dentry (directory entry): Представляет запись в директории, связывает имя файла с inode. Кэшируется для ускорения путей.
- file: Представляет открытый файл, связанный с процессом. Содержит позицию в файле и указатели на операции.

Блочный Input/Output:

- Запросы на чтение/запись от ФС попадают в I/O scheduler, который оптимизирует порядок операций для уменьшения времени позиционирования головок диска (например, CFQ, Deadline, NOOP).
- Оптимизированные запросы объединяются в структуру struct bio и отправляются на уровень драйвера блочного устройства.

2.5. Сетевой стэк в Линуксе

Многоуровневая реализация сетевых протоколов.

Восходящий поток (Receive Path):

1. Сетевой адаптер через прерывание или NAPI (polling) передает пакет в драйвер.
2. Пакет помещается в структуру struct sk_buff (socket buffer) или сокет — основная структура данных для работы с сетевыми пакетами. Сокет - абстракция сетевого стека
3. Обработка на уровне Link (Ethernet), затем Network (IP).
4. Маршрутизация: ядро решает, предназначен ли пакет локальной системе, нужно ли его форвардить или отбросить.
5. На уровне Transport (TCP/UDP) пакет помещается в соответствующий сокет.

todo: отличие TCP, UDP

6. Приложение читает данные из сокета через системный вызов.

Нисходящий поток (Transmit Path): Движение в обратном порядке: сокет -> TCP/UDP -> IP -> Ethernet -> драйвер сетевой карты.

todo: вставитьб картинку из презентации

todo: NetFilter

2.6. Драйверы устройств в архитектуре Линукса

Унифицированная модель устройств: Ядро предоставляет единый каркас для представления устройств, их шин, драйверов и классов.

Типы драйверов:

- Драйверы символьных устройств: Прямой доступ к данным (например, клавиатура, мышь, /dev/random).
- Драйверы блочных устройств: Доступ через буферизацию и кэширование, блоками (например, жесткие диски, SSD).
- Сетевые драйверы: Работают с сетевыми интерфейсами, оперируют пакетами (sk_buff), не имеют файлового представления в /dev.
- Интерфейсы: Каждая подсистема (USB, PCI, I2C) определяет свой стандартный интерфейс, который должен реализовать драйвер.

2.7. Процессы и потоки

Процесс: Инстанс программы с собственными ресурсами (память, файлы).

Поток (Thread): Поток выполнения внутри процесса. Все потоки одного процесса разделяют его ресурсы (память, файловые дескрипторы), но имеют собственные стеки и регистры.

Реализация в Linux: Потоки реализованы как «легковесные процессы» (Lightweight Processes, LWP). Ядро видит и процессы, и потоки как задачи (tasks), представленные struct task_struct. Разница в том, на какие ресурсы они указывают.

TODO: более подробно

2.8. Переключение контекста и миграция задач по ядрам процессора

2.9. Контекст процесса

2.10. Доступ к текущему процессу

2.11. Блокирование и пробуждение

2.12. Вытеснение задач в терминах контекста процесса

2.13. Системный вызов clone()

Низкоуровневый системный вызов для создания нового потока выполнения.

Ключевая особенность: Принимает флаги, которые определяют, какие ресурсы будут разделяться с родительским процессом, а какие будут скопированы/изолированы.

- CLONE_VM: Потоки разделяют адресное пространство.
- CLONE_FILES: Потоки разделяют таблицу файловых дескрипторов.
- CLONE_FS: Потоки разделяют информацию о ФС (root, current directory).

- CLONE_SIGHAND: Потоки разделяют таблицу обработчиков сигналов.

fork() реализован как вызов clone() без флагов разделения.

pthread_create() реализован как вызов clone() с флагами разделения большинства ресурсов.

2.14. Пространства имен и контейнеры

Пространства имен (Namespaces) — механизм изоляции глобальных системных ресурсов для группы процессов. Процессы в разных неймспейсах видят разные наборы ресурсов.

Основные типы неймспейсов:

- PID: Изоляция дерева процессов.
- Network: Изоляция сетевых интерфейсов, таблиц маршрутизации, портов.
- Mount (mnt): Изоляция точек монтирования.
- UTS: Изоляция hostname и domainname.
- IPC: Изоляция объектов System V IPC и POSIX message queues.
- User: Изоляция UID/GID.
- Cgroup: Изоляция корня cgroup.

Контейнеры (Docker, LXC) — это, по сути, изолированная группа процессов, достигнутая за счет комбинации неймспейсов и cgroups (для ограничения ресурсов).

3. Информация о курсе

Поток — y2024.

Группы M3232-M3239.

Преподаватель — Романовский Алексей Валентинович