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. Информация о курсе

Поток — y2024.

Группы M3232-M3239.

Преподаватель — Романовский Алексей Валентинович