Конструирование ядра операционных систем (VIII)

Системные вызовы (Syscalls)

План

- Пользовательское пространство
- Прерывания и системные вызовы
- Дополнительные механизмы защиты памяти
- Динамическое инструментирование

Userspace

- Уровень привилегий определяется младшими битами 1:0 сегментного регистра CS (ring 0 — kernel и ring 3 — user).
- Доступ пользователя к странице в памяти обеспечивается установкой бита U (user) в таблице страниц.
- В непривилегированном режиме запрещены выполнение некоторых инструкций и доступ к отдельным регистрам.
- *Защита ядра от программы в непривилегированном режиме — базовая задача любой многозадачной ОС.*

Kernelspace → Userspace

- Для перехода в пользовательский режим используется инструкция iret (iretq), которая восстанавливает контекст пользовательского адресного пространства из стека.
- Также можно использовать far инструкции (e.g. retf), но они менее удобны из-за ручной работы с контекстом.

Kernelspace ← Userspace

- Для возврата в режим ядра используются прерывания разных видов и/или системные вызовы.
- При обработке прерываний происходит загрузка контекста ядра из TSS (Task State Segment).
- При обработке системных вызовов могут использоваться специальные расширения.

TSS – Task State Segment

Представляет собой некоторую структуру в памяти, которая содержит информацию для перехода в ядро:

- RSP# — стек для перехода на уровень CPL# (0-2).
- IST# — стек для прерываний (1-7), опциональны в IDT.

- Адрес TSS указывается в записи в таблице GDT, на которую указывает регистр TR (аналогично сегментным регистрам).
- Гипотетически позволяет делать аппаратное переключение между задачами, но из-за размера GDT не используется.

Типы прерываний (1/2)

По терминологии x86:

- Синхронные – exception (немаскируемые)
- Асинхронные – interrupt (маскируемые, флаг IF)

Типы прерываний (2/2)

- Синхронные – результат выполнения текущей инструкции
  – int XXh
  – деление на ноль
  – разыменование указателя
    - Страница отсутствует (нет бита P)
    - Страница недоступна на запись/исполнение
- Асинхронные – внешние относительно процессора
  – прерывания таймера
  – от других устройств (e.g. PCI)

IDT — Interrupt Descriptor Table

- 256 слотов, 0-31 зарезервированы Intel
  - 13 – General Protection Fault
  - 14 – Page Fault
- Регистр IDTR содержит адрес таблицы
- Слот в таблице:
  - CS:IP – Kernel Code
  - Ring – уровень привилегий
  - Present – наличие данного слота
  - Gate Type — interrupt (IF=0)/trap (IF=1)
    REF: stackoverflow.com/a/14009799
  - IST — номер стека в TSS

Системные вызовы

- Общий механизм заключается в сохранении аргументов системного вызова в регистры в соответствии с ABI (по аналогии с вызовом функций) и переходом в режим ядра.
- Могут быть реализованы как обычные прерывания с помощью инструкции int. В JOS используется int 30h.
- Для ускорения переключения контекста в современных процессорах реализованы специальные механизмы для реализации системных вызовов.

Ускоренные системные вызовы

- SYSCALL/SYSRET — стандартный механизм системных вызовов для 64-битных приложений.
  - Не использует стек (RSP остаётся неизменным).
  - Загружает RIP из IA32_STAR MSR (RIP сохраняется в RCX).
  - Флаги сбрасываются по IA32_FMASK MSR (например, IF).
- SYSENTER/SYSLEAVE — стандартный механизм системных вызовов для 32-битных приложений.
  - Не использует стек (ESP меняется на IA32_SYSENTER_ESP MSR).
  - Загружает RIP из IA32_SYSENTER_EIP MSR (EIP теряется).
  - Прерывания маскируются безусловно.
- SYSCALL/SYSRET в процессорах AMD в 32-битном режиме.

Дополнительная защита памяти

- Биты R/W и NX в таблице страниц — базовые механизмы предотвращения непреднамеренного доступа к памяти. Не зависят от CPL — пользовательская исполняемая память остаётся исполняемой в ядре.
- SMEP (Supervisor Mode Execution Prevention, см. CR4) запрещает исполнение памяти пользователя (U) в ядре.
- SMAP (Supervisor Mode Access Prevention, см. CR4) запрещает любой доступ к памяти пользователя в ядре. Может быть временно отключено битом AC в RFLAGS.

Динамическое инструментирование

- Анализ программного кода, который выполняется в процессе работы программного обеспечения.
- Инструментирование может выполняться как внутри программного обеспечения, так и в некоторой изолированной среде, например, в виртуальной машине.
- Основные направления развития: поиск ошибок (санитайзеры LLVM, Valgrind, фаззеры, etc) и сбор метрик (покрытие, профилирование, etc).

Санитайзеры LLVM

LLVM — набор технологий для компиляции и сборки кода.

LLVM Sanitizers — набор инструментов от Google для динамического анализа кода, встраиваемый в программу автоматически на этапе компиляции.

—AddressSanitizer
—EfficiencySanitizer
—LeakSanitizer
—libFuzzer
—MemorySanitizer
—SafeStack
—ThreadSanitizer
—UndefinedBehaviorSanitizer

Реализация санитайзеров LLVM

1. Поддержка на уровне компилятора (кодогенерация).
2. Поддержка на уровне среды выполнения (runtime):
   - Адаптация встроенной
   - Адаптация сторонней
   - Разработка собственной
3. Поддержка на уровне исходного кода:
   - Адаптация кода инициализации
   - Адаптация специальных мест (аллокаторов, системных вызовов, обработчиков прерываний, примитивов синхронизации, etc)
   - Усиление инструментов через аннотации кода

Использование санитайзеров LLVM

- UBSan — средство поиска неопределённого поведения в коде на C (арифметика, сдвиги, работа с битами).
- ASan — средство поиска ошибок работы с памятью (use-after-free, buffer overflow/underflow, out of scope usage).
- MSan — средство поиска неинициализированной памяти (при ветвлении, передаче в системные вызовы).
- TSan — средство поиска гонок в многопоточном коде.

REF: https://github.com/google/sanitizers

Фаззинг

- Способ тестирования ПО, заключающийся в генерации псевдослучайных входных данных, с целью вызвать «интересное» поведение программы.
- Фаззеры классифицируются по:
  - Знанию структуры входных данных: «умный» или «глупый»
  - Знанию структуры ПО: «белого» или «чёрного» ящика
  - Построению входных данных: генерационный, мутационный и эволюционирующий (на основе обратной связи)
- Фаззеры дополняют динамический анализ (e.g. libFuzzer).

REF: Lab8-FUZZ.pdf

Спасибо за внимание!

Вопросы?