Семинар 11: Intel x86 assembly. Часть 4

11 февраля, 2020

Материалы семинаров на GitHub

Системные вызовы

- Способ общения с операционной системой
- В C выглядят как обычные функции
- Реализуются libc
- В ассемблере всё хитрее

Системные вызовы: механизм работы

- Все программы работают в непривилегированном режиме (ring3)
- В этом режиме нельзя напрямую работать с физической памятью, устройствами и портами ввода-вывода
- Как перейти в привилегированный режим (ring0)?

Прерывания процессора

- Прерывания — специальные события, которые «прерывают» исполнение команд
- Прерывание может произойти после любой инструкции, но не во время её выполнения
- Interrupt vector table — массив указателей на функции-обработчики прерываний (x86)

Прерывания процессора

- Hardware interrupts — железные события
- Software interrupts — события, сгенерированные программами
- Exceptions — исключительные ситуации процессора

Примеры прерываний

- События сетевой карты, клавиатуры, видеокарты, ...
- `int` и `syscall`
- Divide-by-Zero
- Page fault (double fault, triple fault)
- General Protection Fault
- Invalid Opcode
- x87 Floating-Point Exception
- SIMD Floating-Point Exception

Системные вызовы: x86

- Порядковый номер вызова записывается в **eax**
- Аргументы передаются в **регистрах**: **ebx**, **ecx**, **edx**, **esi**, **edi**
- Затем делается `int 0x80`

Таблица системных вызовов Linux x86

Системные вызовы: x86-64

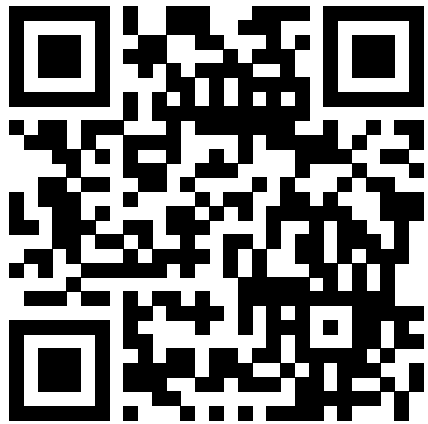
- Порядковый номер вызова записывается в **rax**
- Аргументы передаются в **регистрах**: **rdi**, **rsi**, **rdx**, **r10**, **r8**, **r9**
- Затем делается `syscall`

Таблица системных вызовов Linux x86-64

Red zone

- Область размером 128 байт **под** текущим **rsp**
- System V AMD64 ABI гарантирует, что это место не используется обработчиком прерывания
- Используется для оптимизации, например, последняя функция может не выделять полноценный фрейм, а использовать red zone

Что будет, если забыть про red zone?



Файлы

- Каждый файл имеет путь относительно /
- Передавать каждый раз путь в системный вызов – затратно
- Поэтому (и не только) изобрели *файловые дескрипторы*
- Файловый дескриптор – какое-то число, которое идентифицирует файл

Правило #1: всегда закрывать файловые дескрипторы, после завершения работы с ними!

Файловые дескрипторы

- 0 – stdin, 1 – stdout, 2 – stderr
- Выдаются последовательно
- Под файловым дескриптором может быть что угодно: регулярные файлы, именованные каналы, сокеты, linux-specific интерфейсы (epoll, signalfd, eventfd), ...
- Ограниченное количество, лимит задаётся на каждый процесс, увеличить можно с помощью ulimit
- Утекание файловых дескрипторов намного страшнее утечек памяти

Интерфейс Linux для файлов

```c
#include <unistd.h>
#include <fcntl.h>
int open(const char *pathname, int oflag, mode_t mode);
ssize_t read(int fd, void *buf, size_t nbytes);
ssize_t write(int fd, const void *buf, size_t nbytes);
int close(int fd);
```

open

- «Открывает» файл с заданным режимом (на чтение/запись) и возвращает файловый дескриптор
- **O_RDONLY** – только чтение
- **O_WRONLY** – только запись
- **O_RDWR** – и чтение, и запись
- **O_CREAT** – создать файл, если он не существует (mode в этом случае – права на файл)
- **O_EXCL** – используется вместе **O_CREAT** и тогда гарантирует атомарность создания файла

read

- Копирует **ИЗ** файлового дескриптора **fd** *не более* **nbytes** в буфер **buf**
- Возвращает 0, если больше нечего «читать»
- Если произошла ошибка возвращает -1 и выставляет **errno**
- В противном случае заблокируется, пока данные не будут доступны на чтение
- Возвратит количество прочитанных байт

write

- Копирует **в** файловый дескриптор **fd** *не более* **nbytes** из буфера **buf**
- Возвращает -1, если произошла ошибка
- Возвращает количество записанных байт (всегда >0)
- Может заблокироваться, если файл под файловым дескриптором пока нельзя что-либо записать

close

- Закрывает (освобождает) файловый дескриптор
- Возвращает -1, если произошла ошибка – это бывает очень редко

Inline assembly

- Позволяют встраивать код на ассемблере внутрь C/C++
- Используют подстановки в специальном синтаксисе
- Обычно используются только в крайнем случае, когда нужно вставить особую процессорную инструкцию в тело функции (например, инструкцию, которая запрещает прерывания)

Как это ~~отвратительно~~ выглядит

```
asm ("mov %3, %0\n"
     "mov %2, %1\n"
     : "=r" (a), "=m"(b)      // выходные параметры
     : "r" (c), "m"(d)        // входные параметры
     : "rax", "rbx", "rdx")   // clobbers
```

Мануал по inline assembly

Дзякуй!