

Процессы

Software interrupts

- Ещё один вид прерываний, вызываемые специальными инструкциями
- `int X`, `int3`, `into`, `int1`


CPL, DPL и RPL

- CPL (current privilege level) – текущий уровень привилегий, определяется регистром `cs`
- DPL (descriptor privilege level) – уровень привилегий, откуда доступен сегмент (например, DPL в IDT определяет уровень привилегий для софтверных прерываний)
- RPL (requested privilege level) — то, что записывается в нижние биты segment selector

TSS

- Task-state segment
- В protected mode может использоваться для hardware multitasking
- В 64-bit mode используется для хранения информации
- В современных ОС требуется заводить по одному TSS на ядро процессора

31	15	0	
I/O Map Base Address		Reserved	100
Reserved			96
Reserved			92
IST7 (upper 32 bits)			88
IST7 (lower 32 bits)			84
IST6 (upper 32 bits)			80
IST6 (lower 32 bits)			76
IST5 (upper 32 bits)			72
IST5 (lower 32 bits)			68
IST4 (upper 32 bits)			64
IST4 (lower 32 bits)			60
IST3 (upper 32 bits)			56
IST3 (lower 32 bits)			52
IST2 (upper 32 bits)			48
IST2 (lower 32 bits)			44
IST1 (upper 32 bits)			40
IST1 (lower 32 bits)			36
Reserved			32
Reserved			28
RSP2 (upper 32 bits)			24
RSP2 (lower 32 bits)			20
RSP1 (upper 32 bits)			16
RSP1 (lower 32 bits)			12
RSP0 (upper 32 bits)			8
RSP0 (lower 32 bits)			4
Reserved			0

 Reserved bits. Set to 0.

Stack switching

- Если при входе в прерывание меняется CPL, то процессор изменяет RSP на соответствующий новому CPL в TSS
- `RSP0` для ring0, `RSP1` для ring1, итд
- Если CPL не изменяется, то RSP не меняется и фрейм прерывания кладётся прямо на текущий стек

Interrupt stack table

- IST появился впервые в 64-bit mode
- Позволяет прерываниям назначать отдельный стек при входе (переключается безусловно)
- Какой стек выбрать описывается в IDT (3 бита)
- 0 зарезервирован — обозначает то, что прерывание должно использовать старый механизм переключения стека

Процессы, системные вызовы и переключение контекста

Что такое процесс?

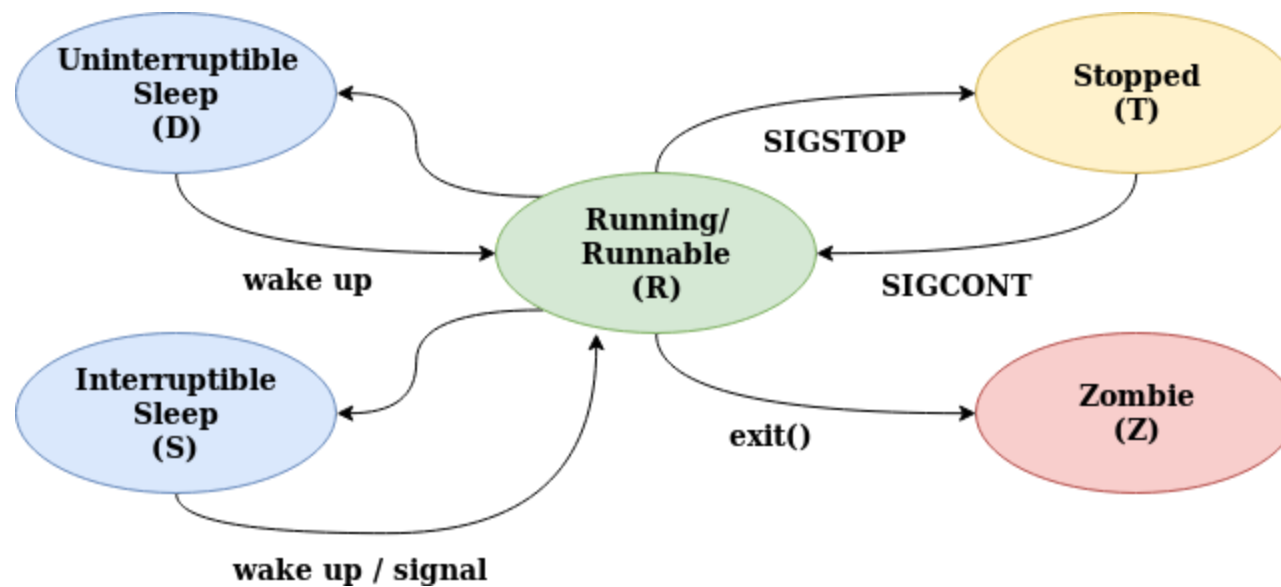
Что такое процесс?

- Процесс — это «запущенная программа», исполняемая процессором
- Результат загрузки программы из исполняемого файла называется *образом* (image)
- Контекст исполнения: RIP, EFLAGS, RSP, etc
- Адресное пространство
- Ресурсы ОС: файловые дескрипторы, сокеты итд

Процессы и треды

- Внутри процесса может быть несколько потоков
- Удобно представлять процесс как группу тредов, разделяющих одно адресное пространство и некоторые другие ресурсы
- В Linux треды называются тасками и описываются `struct task_struct`

Состояния процесса



User-space и kernel-space

- После перехода в обработчик системного вызова, процесс находится в kernel-space
- Ядро использует выделенный per-task стек небольшого размера — kernel stack
- В kernel-space тоже могут возникать прерывания, поэтому в критических местах их нужно отключать

Системные вызовы

- Процессы изолированы от периферии, поэтому нужен способ взаимодействия с операционной системой
- Системные вызовы — это API для ОС

Системные вызовы: `int 0x80`

- 32-bit linux для системных вызовов использует программное прерывание 128 (0x80)
- В `eax` передаётся номер системного вызова
- Аргументы передаются через регистры (по порядку): `ebx` , `ecx` , `edx` , `esi` , `edi`
- Прерывания — медленный механизм: проверки на стороне процесса

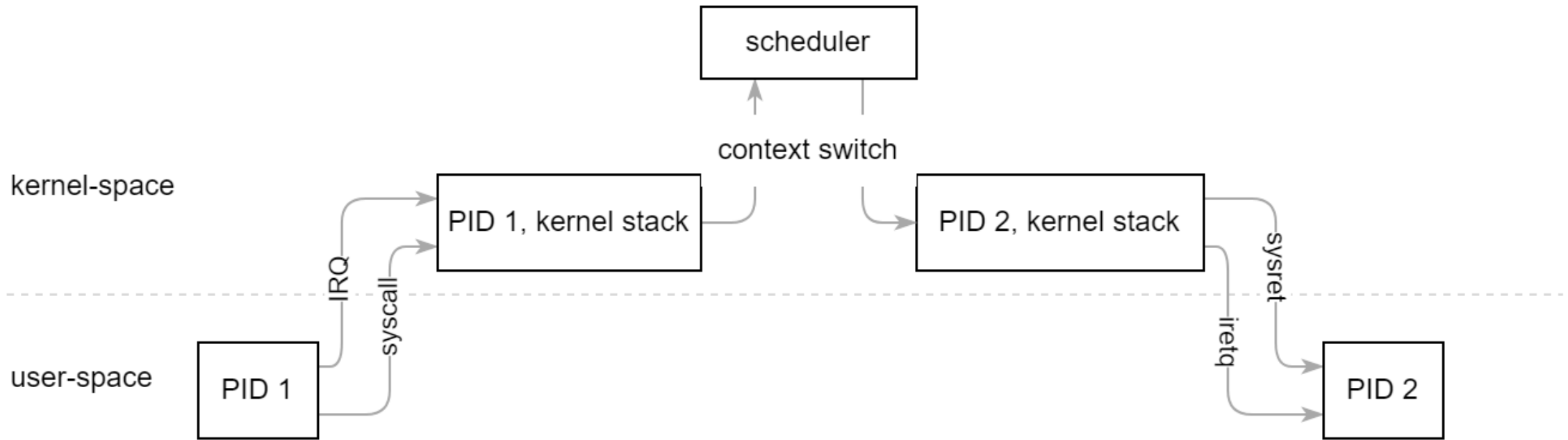
Системные вызовы: `sysenter/sysexit`

- Вариант Intel
- AMD поддерживает только в 32 bit mode
- `IA32_SYSENTER_CS` задаёт описание «виртуального» code segment: он не описывается в GDT/LDT, а имеет константные значения
- `IA32_SYSENTER_ESP` — стек при заходе в ядро
- `IA32_SYSENTER_EIP` — адрес точки входа
- Не является частью ABI ядра

Системные вызовы: `syscall/sysret`

- Изначально придуман AMD
- Intel поддерживает только в 64 bit mode
- `IA32_LSTAR` — адрес точки входа
- `IA32_STAR` аналогично `IA32_SYSENTER_CS`
- `rip -> rcx`, `rflags -> r11`
- Аргументы передаются через регистры (по порядку): `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9`

Переключение контекста



Переключение контекста: `schedule`

```
void schedule() {  
    // called from task A  
    struct task* next = select_next_task();  
    switch_context(current, &next);  
    // already in task B  
    vmem_switch_to(next->vmem);  
    restore_fpu(next->fpu_ctx);  
    // ...  
}
```

Переключение контекста: `switch_context`

- Основная магия
- Переключает процесс на другой ядерный стек
- Можно не сохранять все регистры, только callee-saved, т.к. компилятор знает, что вызывает функцию, => уже сохранит нужный ему контекст

Переключение контекста: `switch_context`

```
// rdi points to current context
// rsi points to next context
switch_context:
    push rbp
    push rbx
    // ...

    // save current stack
    mov qword ptr [rdi], rsp

    // restore next stack
    mov rsp, qword ptr [rsi]

    // ...
    pop rbx
    pop rbp
    ret
```

На этом всё!