Операционные Системы

Системные вызовы

May 11, 2017

Системные вызовы

- Системные вызовы - интерфейс между userspace и ядром ОС
  - пользовательский код не имеет достаточно привилегий, чтобы вызывать код ядра как обычные функции;
  - системный вызов сопровождается повышением привилегий;
  - возврат из системного вызова сопровождается понижением привилегий.

Реализация системных вызовов

- Как реализовать интерфейс системных вызовов?
  - способ, как обычно, зависит от архитектуры;
  - например, в x86 существуют инструкции syscall и sysenter;
  - но мы посмотрим на другой вариант (более старый).

И сново о прерываниях...

- Что происходит, если обработчик прерывания прерывает пользовательский код?
  - вызывается обработчик прерывания - код ядра;
  - обработчик прерывания выполняется уже в привилегированном режиме.

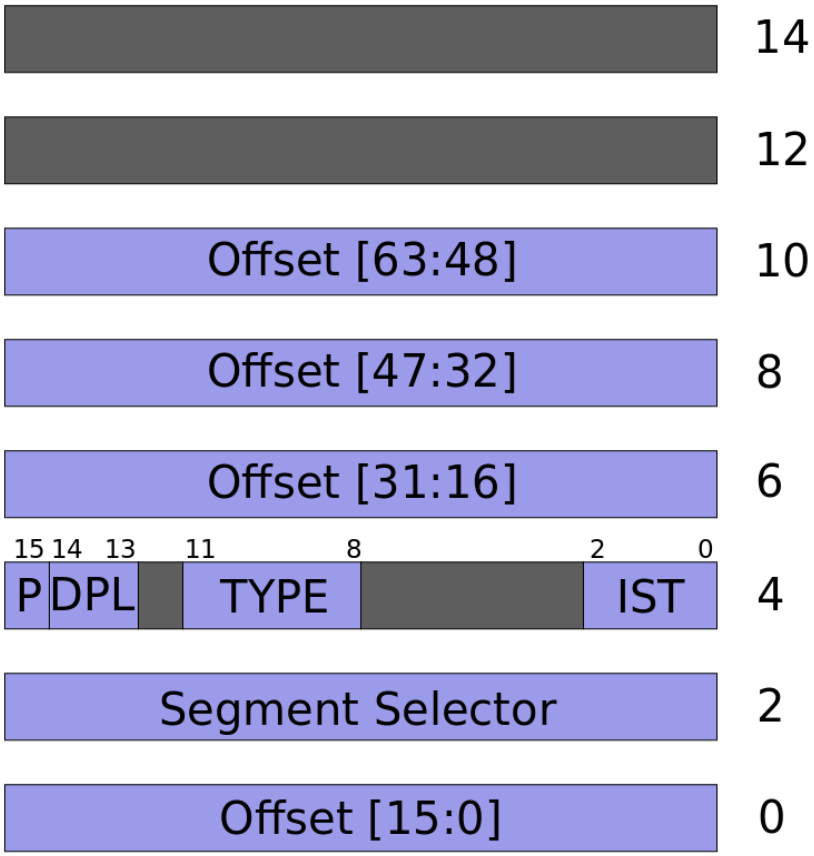
Программные прерывания

- Прерывания можно вызывать программно (и не только сделав ошибку)
  - например, в x86 для этого существует специальная инструкция int, номер прерывания - параметр инструкции;
  - выберем запись в IDT и будем использовать ее для системных вызовов.

Программные прерывания в x86

- С помощью инструкции int в x86 можно генерировать прерывание с любым номером
  - в том числе и соответствующие исключениям;
  - в том числе и соответствующие аппаратным прерываниям;
  - приложения могут натворить бед, если разрешить им генерировать прерывания как попало.

Дескриптор IDT, поле DPL



Дескриптор IDT, поле DPL

- DPL дескриптора системного вызова выставляем в 3
  - благодаря чему непривилегированный код может генерировать это прерывание.
- DPL всех остальных дескрипторов выставляем в 0.

Стек обработчика прерывания

- При вызове обработчика на стек сохраняются адрес возврата и прочее
  - на какой стек все это будет сохранено?
  - не хочется использовать стек непривилегированного кода
    - там может быть не достаточно места;
    - пользовательский код может делать со своим стеком все что угодно.

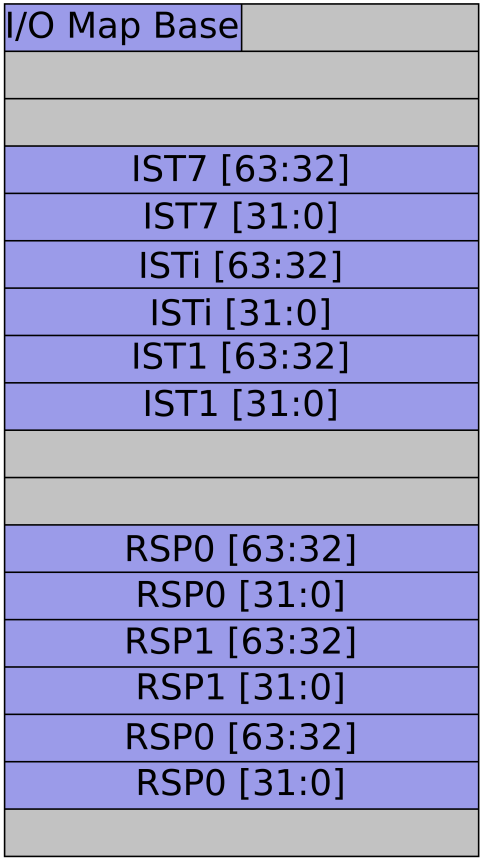
Отдельный стек для ядра

- Мы хотим использовать отдельный стек для ядра и отдельный для userspace
  - например, в Linux для каждого потока создается стек ядра, т. е. у каждого потока есть 2 стека;
  - при прерываниях и системных вызовах происходит переключение на стек ядра потока.

Task State Segment

- TSS (Task State Segment) - структура, которая хранит указатель стека, который будет загружен в RSP
  - ранее (в 32-битном режиме) могла быть использована для хранения состояния потока.

"Прыжок" в userspace



Task State Segment

- "Указание" на TSS хранится в специальном регистре TR
  - инструкция LTR записывает значение в TR, а инструкция STR читает;
  - для использования TSS необходимо завести специальный дескриптор в GDT
    - Base и Limit хранят логический адрес и размер TSS;
  - селектор дескриптора сохраняется в TR.

Task State Segment

- Простой вариант использования TSS:
  - создаем по TSS на каждое ядро процессора - один раз, при инициализации ядра ОС
  - при переключении потоков подменяем указатель стека в TSS.

Резюме

- Подготовить дескриптор IDT, который будет использоваться для системных вызовов.
- Создать TSS:
  - создать дескриптор, описывающий TSS, в GDT;
  - загрузить селектор, ссылающийся на дескриптор, в TR.
- Не забывать подменять указатель стека в TSS при переключении потоков.

"Прыжок" в userspace

- Как передать управление в userspace в первый раз?
  - инструкция iretq завершает обработчик прерывания и передает управление, возможно, понизив уровень привилегий;
  - инструкция iretq берет свои параметры со стека - подготовим стек и вызовем iretq.

"Прыжок" в userspace