Interrupts

Me

September 12, 2016

Contents

1 Введение

Когда внешнее устройство генерирует сигнал, происходит исключительная ситуация или исполняется специальная команда генерация программного прерывания процессор должен прервать исполнение текущей задачи и вызвать обработчик прерывания [1].

Для исключительных ситуаций используются первые 32 прерывания[2], начиная с 32 прерывания до 256 не включительно, они доступны для произвольного использования.

Код, на который процессор должен переключиться при получении прерывания, называтеся обработчиком прерывания. В этом разделе мы каснемся того, как сообщить процессору какой обработчик и когда вызывать, как должен выглядеть обработчик прерывания, как запрещать/разрешать прерывания на процессоре.

2 Таблица дескрипторов прерываний

Таблица дескрипторов прерываний (так же известная как IDT) указывает процессору какие обработчики прерываний за какие прерывания отвечают. Кроме того IDT определяет, когда можно вызывать обработчик прерывания и в какой "режим" должен перейти процессор перед тем как вызвать обработчик.

Указатель на IDT хранится в специальном регистре IDTR. Обращаться к этому регистру с помощью обычных операций вроде mov нельзя, но для работы с ним используются специальные инструкции lidt и sidt:

- инструкция lidt записывает значение в IDTR;
- sidt считаывает значение из IDTR.

Значение в регистре IDTR имеет специальный вид:

```
1 struct idt_ptr {
2         uint16_t limit;
3         uint64_t base;
4 } __attribute__((packed));
```

[1] Обатите внимание, мы пока не касаемся каким именно образом внешние устройства посылают процессору сигнал, нас пока интересует только та часть общей картины происходящего, за которую отвечает непосредственно процессор.

[2] Реально используются только часть из них, неиспользуемые считаются зарезервированными.

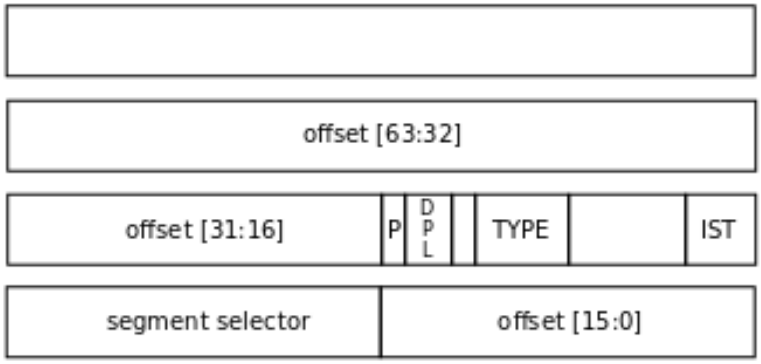


Figure 1: Запись в таблице IDT.

- limit - размер таблицы в байтах минус 1;
- base - адрес таблицы[1].

Теперь касательно самой IDT. Таблица состоит из дескрипторов фиксированного формата. На каждый из 256 возможных прерываний по одному дескриптору. Обратите внимание, что размер таблицы максимум 256 записей, но никто не запрещает использовать таблицу и меньшего размера. Главное следить за тем, чтобы не произошло прерывание, для которого нет соответствующей записи в таблице.

Каждая запись в IDT состоит из 16 байт и имеет вид показанный на Рис. **??**.

- offset - самое очевидное поле, оно указывает на адрес начала обработчика прерывания[2];
- segment selector - значение, которое будет загружено в сегментный регистр CS перед вызовом обработчика прерывания;
- P - бит присутствия, если он установлен, то запис воспринимается как валидная, в противном случае она невалидная и при получения прерывания для такой записи в IDT произойдет ошибка[3];

[1] Я использую нотацию gcc, аттрибут packed, значит, что компилятор не будет вставлять никаких padding-ов чтобы добиться подходящего выравнивания для полей структуры.

[2] Смотрите на этот адрес как на указатель на функцию обработчик.

[3] Еще одно прерывание.

- DPL (Descriptor Priviledge Level) - это поле используется для защиты обработчика прерывания от вызова в некоторых обстоятельствах;
- TYPE - дескрипторы бывают нескольких типов, нас, пожалуй, могут заинтересовать только два - Interrupt Gate (значение 14) и Trap Gate (значение 15);
- IST - поле, которое отвечает за переключение стека при вызове прерываний, это полезная вещь с точки зрения безопасности, но нас она особо не будет интересовать, так что можно спокойно задавать это поле равным 0;
- все остальные поля зарезервированы и должны быть равны 0.

2.1 Descriptor Priviledge Level

Как вы возможно знаете, прерывания могут приходить не только от внешних устройств, но также как реакция на какие-то ошибочные[1] ситуация или генерироваться специальными инструкциями, например, инструкцией int.

Инструкция int как аргумент принимает номер прерывания, обработчик которого нужно вызвать. Это может создавать проблемы, например, представим, что прерывание номер x настроено на работу с каким-нибудь внешним устройством, например, сетевой картой. Пользовательское приложение может саботировать работу системы намерено генерируя инструкцию int с аргументом x. Зачастую это неприемлимо для ОС возволять какому-то приложению испортить жизнь всем остальным приложениям. В конце концов, одна из задач ОС - изолировать приложения друг от друга, чтобы ни одно из них не могло помешать другим[2].

Так вот, поле DPL указывает уровень привелегий, которым должен обладать код [3], чтобы он мог сгенерировать прерывание, например, инструкцией int. Конечно, если прерывание сгенерировано устройством, то эта проверка, конечно, не имеет смысла так как мы не можем делать никаких предположений касательно кода исполняемого на процессоре

[1]Тут, забегая далеко вперед, я должен сказать, что эти ситуация далеко не всегда являются неожиданными ошибками, иногда ОС намернно ожидает исключения и они могут стать частью логики работы ОС.

[2]Стоит отметить, что когда-то была довольно популярная ОС DOS, которая не изолировала приложения друг от друга, так что все это довольно условно.

[3]Два младших бита регистра CS.

в момент получения сигнала от устройства, соответственно, и это поле будет проигнорировано в этом случае.

Таким образом для всех аппаратных прерываний это поле должно иметь значение 0, а, например, для прерыываний отведенных под системные вызовы это поле должно иметь значение 3.

2.2 Segment selector

Назначение этого поля, на первый взгляд, очевидно - оно будет записано в регистр CS. Но в чем назначение регистра CS? В 64-битном режиме x86 (он же Long Mode), основное назначение - определять уровень привелегий[1]. Уровень привилегий кода определяется двумя младшими битами значения регистра CS (0 - наибольший уровень привелегий, а 3 - наименьший).

Приключения начинаются, если уровень привилегий прерываемого кода, не совпадает с уровнем привилегий обработчика прерываний, т. е. в ситуации если процессору нужно сменить уровень привилегий, особенно, если мы переходим с более низкого уровня привилегий к более высокому, например, от уровня 3 к уровню 0. И главная проблема - это стек.

Стек является очень важным регионом памяти. На стеке зачастую храняться локальные переменные вашего кода, на стек сохраняют адреса возврата, чтобы было понятно куда возвращать управление после завершения функции. Другими словами, стек во многом определяет работу вашего кода. Кроме всего прочего, в стек будет сохранена информация о коде прерванном при вызове обработчика прерывания [2].

А теперь представим ситуацию, когда обработчик прерывания прерывает непривилгированный код, управление передается обработчику прерывания, т. е. ядру ОС и ядро начинает сохранять на стек свою информацию. Но рано или поздно обработчик прерывания завершится, и управление будет возвращено прерванному коду, который в свою очередь может посмотреть, что же находится на стеке и, таким образом, получить информацию из ядра ОС.

Таким образом, использовать стек пользовательского приложения

[1] Что не значит, что не нужно иметь соответсвующую запись в таблице GDT, даже если большая часть полей

[2] Интересный момент, в архитектуре ARM адрес возврата сохраняется не в стек, а в специальны регистр, после чего его можно перенести куда угодно на ваше усмотрение, впрочем, зачастую этот адрес всеравно попадет в стек, если конечно вам вообще понадобится сохранить его куда-то еще.

для кода ядра ОС идея не самая светлая[1], а процессор должен переключиться на использование новго стека специально для ядра ОС. Поле IST как раз отвечает за это, но не только оно. Однако на данной стадии нам не нужно об этом беспокоится по той простой причине, что у нас, пока, есть только привелигированный код, а описанная выше проблема нас не волнует.

2.3 Маскировка прерываний и поле type

Процессору можно сказать, чтобы он не принимал прерываний от внешних устройств. Обратите внимание речь только о внешних устройствах, не отреагировать на ошибочную ситуацию или на явный запрос от пользователя было бы очень странно.

Разрешено или нет процессору принимать прерывания определяется специальным битом IF (номер 9 считая с 0) в флаговом регистре RFLAGS. Чтобы изменить значение этого флага можно воспользоваться, например, инструкциями cli и sti.

Инструкция cli очищает этот флаг и запрещает процессору принимать прерывания, а sti - обратная инструкция, которая устанавливает этот флаг и, соответсвенно, разрешает прерывания[2].

Как уже отмечалось, поле type нас в основном интересует для того чтобы выбрать между Interrupt Gate и Trap Gate. В чем между ними разница? В случае Interrupt Gate флаг IF будет очищен перед вызовом прерывания, т. е. прерывания от внешних устройств будут запрещены. Trap Gate не запрещает прерывания, и, соответственно, возможна ситуация когда один обработчик прерывания будет прерван другим обработчиком прерывания, что само по себе не является проблемой.

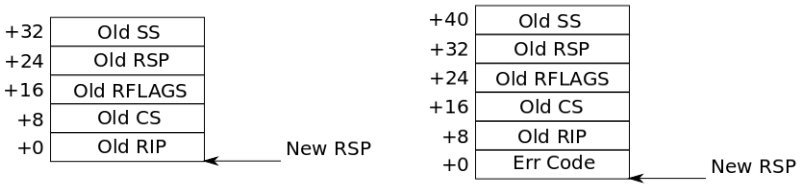
2.4 Обработчик прерывания

Итак, offset указывает на начало обработчика прерывания, как должен выглядеть этот обработчик прерывания. Перед тем как разбираться как должен выглядеть обработчик прерывания, нам стоит понять в каком состоянии находится процессор, когда обработчик получает управление.

Мы точно знаем, что в регистре CS содержится значение segment selector из соответствующей записи в IDT, а в регистре RIP адрес обработчика прерывания, записанный в поле offset записи IDT. Иногда мы так же можем знать один бит из регистра RFLAGS, но это не так важно в данный момент.

[1]Впрочем, ядро ОС вполне может почистить стек за собой.

[2]Не трудно запомнить, cli - clear interrupt flag, a sti - set interrupt flag.



(a) Stack without error code.

(b) Stack with error code.

Figure 2: Interrupt handler stack frame.

Более существенно то, что находится на вершине стека. На Рис. 2a и Рис. 2b показаны два варианта состояния стека. Для внешних устройств и программных прерываний возможен только первый вариант[1], в то время как для исключений возможны оба, какой именно используется определяется номером исключения. Вариант показанный на Рис. 2b используется для исключений с номерами 8, 10-14 и 17.

Обратите внимание, что о состоянии преравнного кода нам изместно только значение регистров CS, SS, RIP, RSP и RFLAGS. Где же хранится все остальное? На самом деле нигде - все остальное вы должны сохранить самостоятельно. Другими словами все остальные регистры, если вы собираетесь их менять в обработчике прерывания вы должны самостоятельно сохранить, например, в стек. Восстановить сохраненную информацию, опять же, ваша задача.

Другой важный момент - завершение обработчика прерывания. Обычно обработчик прерывания завершается инструкцией iretq[2]. Инструкция iretq ожидает, что на врешине стека будет находится информация показанная на Рис. 2a. Причем даже если при вызове обработчика в стек был помещен код ошибки инструкция iretq ожидать его не будет, и это ваша задача удалить его со стека.

Что делает инструкция iretq? Она берет информацию со стека и распихивает ее по соответствующим регистрам - ничего особенного. Но из этого следует, в частности, что вам не обязательно пользоваться инструкцией iretq, т. е. если вы сможете повторить все тоже самое, что делает iretq без, собственно, iretq то это тоже будет работать. И наоборот, нет никакого запрета на использование iretq за пределами обработчика прерываний, если вам нужно то, что она делает.

Кроме того, для внешних прерываний, вам обычно нужно сообщить

[1]Инструкцией int мы можем сгенерировать прерывание с любым номером, но стек всегда будет выглядеть одинаково.

[2]На самом деле инструкция iret, но у нее есть несколько вариаций, и iretq - та из них, которая нужна нам в 64-битном режиме работы.

контроллеру прерываний о том, что вы обработали прерывание, но эта часть истории будет рассказана в другом разделе.

2.5 Выводы

Прочитав эту часть информации вы должны суметь создать IDT, уметь запрещать и разрешать прием прерываний и писать обработчики прерываний. Пока вы не знаете как общаться с контроллером прерываний вы можете попрактиковаться на исключениях. В частности, исключение деления на 0 очень легко сгенерировать (делению на 0 соответсвует 0-ая запись в IDT[1]), а также вы можете воспользоваться командой int.

Если вы будете практиковаться, пожалуйста обратите внимание какое значение RIP попадает на стек в случае, если происходит деление на 0 и в случае использования инструкции int.

3 Контроллер прерываний

Данный раздел относится исключительно к прерываниям от внешних устройств, и рассказывает ту часть истории, которая происходит снаружи CPU[2]. К контроллеру прерываний подключаются устройства, которые хотят генерировать прерывания, а контроллер прерываний уже отправляет сообщения процессору. В сообщении каким-то образом сообщается какой номер прерывания (какую запись IDT) использовать для вызова обработчика прерывания.

Контроллер преываний выполняет арбитраж, т. е. выстраивает запросы от устройств в очередь, согласно некоторому приоритету. Впрочем детали приоритизации запросов от устройств нас не особо волнуют на данном уровне понимания. Зато начальная настройка контроллера прерываний, отображение на записи IDT нас очень волнуют.

Мы будем использовать для экспериментов устаревший контроллер прерываний из серии intel 8259. Каскад из двух таких чипов использовался в IBM PC начиная с версии AT, т. е. в предке того, что мы сейчас называем персональным компьютером. Зачастую современные персональные компьютеры сохраняют обратную совместимость, т. е. поведение этих intel 8259 чипов эмулируется современными

[1]Полный список исключений вы можете найти в документации Intel.

[2]Впрочем снаружи CPU в данном случае мало что значит, контроллер прерываний с тем же успехом может находится и прямо в CPU.

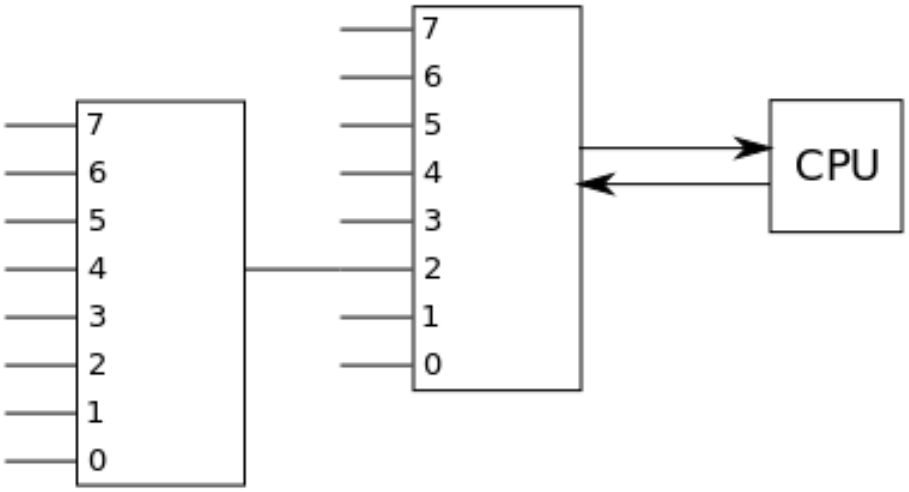


Figure 3: Каскад intel 8259 в IBM PC AT.

контроллерами прерываний используемыми в персональных компьютерах. Соответственно, вся конкретная информация относится только к intel 8259, в то время как общая картина меняется с изменением контроллера не так сильно.

3.1 Каскад контроллеров прерываний

Как уже отмечалось, в IBM PC два intel 8259 чипа объединялись в каскад. Конфигурация этого каскада всегда фиксированная, но вообще intel 8259 позволял использовать различные конфигурации, например, объединять больше двух контроллеров.

В каскаде есть Master и Slave (или несколько Slave-ов). Можете считать, что Master контроллер подключается к процессору, в то время как Slave контроллер подключается к Master-у. Устройства могут быть подключены как к Slave контроллеру, так и к Master контроллеру, если у него есть свободные ноги (не занятые подключением Slave-ов). Однако надо понимать, что если устройство подключено к Slave, то Slave потом передаст сигнал Master-у, т. е. на Slave можно очень отдальенно смотреть как на устройство подключенное к Master.

Итак нас интересует конкретно конфигурация каскада для IBM PC. В этой конфигурации единственный Slave подключается ко 2-му выходу Master, считая с 0. Очень схематично эта конфигурация изображена на Рис. 3.

3.2 Взаимодействие с контроллером прерываний

Для общения с контроллером прерываний используется специальная фича архитектуры x86 - пространство ввода/вывода[1]. Для обращения к отдельным местам пространства ввода/вывода используются специальные инструкции in и out. Не трудно догадаться, что in читает что-то из некоторого адреса в пространстве ввода/вывода, а out записывает. Зачастую отдельные адреса пространства ввода/вывода называют портами, далее мы будем придерживаться этого же имени.

Чтобы настроить контроллер прерываний нужно знать какие порты используются контроллером для настройки и что значат отдельные записи в эти порты, ну и конечно чего вы хотите от контроллера прерываний. Оставшаяся часть раздела будет посвещенна именно этому.

Для общения с каждым из двух контроллеров используются свои порты ввода/вывода. Для Master-а используется порты 0x20 и 0x21, а для Slave-а используются порты 0xA0 и 0xA1. Все порты размером в 1 8-ми битный байт. Среди двух портов выделяют командный и порт для данных. Например, 0x20 командный порт для Master, в то время как 0xA0 командный порт для Slave. В командный порт нужно записать некоторое командное слово, после чего аргументы нужные этой команде нужно записать в порт данных и так для каждого контроллера.

3.3 Начальная конфигурация контроллера прерываний

Цель конфигурации контроллера для нас, это задать отображение входов контроллера прерываний на записи в IDT. К сожалению, по крайней мере изначально, intel 8259 создавался как более или менее универасльное устройство, которое будет использоваться не только в IBM PC AT. Поэтому начальная конфигурация задает чуть больше параметров, чем просто отображение входов контроллера на записи IDT.

Каждый раз, когда в командный регистр записывается слово в установленным 4-м битом, это воспринимателся как команда инициализации. Остальные интересные нам биты отвечают за:

- IC4 (бит 0) - если установлен, то команда инициализации ожидает 3 байта данных, если сброшен, то только 2; мы рассмотрим подробнее, зачем нужен каждый байт, сейчас вам достаточно знать, что нам нужны все 3;

[1]Я знаю не так много архитектур, где кроме пространства памяти, есть еще и отдельное пространство ввода вывода. Другая известная мне архитектура это AVR.

- SNGL (бит 1) - если установлен, то используется только один контроллер, в противном случае они объединены в каскад[1];

- LTIM (бит 3) - если установлен, то контроллер прерываний будет ожидать level triggered сигналы от устройств, в противном случае edge triggered; то какой вариант использовать зависит от устройства, в нашем случае всегда используется edge triggered прерывания.

Т. е. в дополнение к 4 биту, в байте, который нужно записать в командный порт каждого из контроллеров, нужно установить бит 0.

Далее оба контроллера будут ожидать 3 байта данных, которые нужно поочереди записать в порт данных каждого из контроллеров. Рассмотрим на каждое из них.

Первый байт, который нужно записать в порт данныз для команды инициализации, это номер 1-ой записи в IDT соответсвующей 0-ому входу контроллера прерываний, т. е. собственно этот байт и задает нужное нам отображение. Для каждого из контроллеров это отображение задается отдельно и так как вам удобно, до тех пор пока вы соблюдаете пару простых правил:

- номера записей в IDT, на которые вы отображаете входы контроллера, должны быть больше или равны 32 (первые 32 записи начиная с 0 зарезервированны);

- не стоит делать так, чтобы отображения разных котроллеров пересекались.

Т. е. если вы хотите отобразить входы котроллеров прерываний на номера с 32, то в порт данных Master нужно записать 32, а в порт данных Slave нужно записать 40.

Второй байт определяет конфигурацию каскада и имеет немного разное значение для Master и Slave. Для Master нужно в этом байте каждый бит соответсвует входу контроллера (их всего 8). Для каждого входа к которому подключен Slave нужно установить 1 в соответствующем бите. В нашем случае 2-ой бит (считая с 0) должен быть равен 1, а остальные сброшены.

Для Slave нужно записать к какому входу Master подключен Slave, на этот раз просто числом, а не в виде битовой маски. Т. е. в нашем случае нужно записать число 2.

[1]Да, контроллерам нужно сообщать, что они объединены в каскад.

Последний байт отвечает за целый набор разных функций, большинство из которых нам не нужны. Из всего байта нам важен только 0 бит, он должен быть установлен в 1 для x86 архитектуры[1].

3.4 End of interrupt

Когда обработчик прерывания завершил обработку, он должен сообщить контроллеру прерываний, что обработка закончена, таким образом контроллер прерываний будет знать, что пара генерировать следующее прерывание, если какое-то из устройств этого попросило[2].

EOI бывает двух разных видов: направленный (Specific) и ненаправленный (Non-Specific). Нам подойдет любой из них. Направленный EOI включает номер ноги, подтверждение для которой мы хотим послать, ненаправленный определяет ее автоматически[3].

Для отправки EOI опять же нужно записать байт в командный регистр. Для направленного EOI биты 5 и 6 (считая с 0) должны быть равны 0, а младшие 3 бита должны содержать номер входа. Для ненаправленного EOI только бит 5 должен быть установлен в 1, а остальные сброшены в 0.

Как уже отмечалось вы можете использовать любой вариант - это не существенно для нас. А вот что важно помнить, так это то, что у нас используется каскад из двух контроллеров прерываний, и если прерывание сгенерировано устройством подключенным к Slave, то Slave дополнительно пошлет сигнал на Master, и, соответственно, послать EOI нужна обоим контроллерам.

Кроме того обратите внимание, что исключения и программные прерывания никакого отношения к контроллеру прерываний не имеют, и для таких прерываний никакого EOI посылать не нужно.

3.5 Маскировка прерываний

Мы можем замаскировать отдельные входы контроллера прерываний. Это делается очень просто и для этого нам даже не нужно писать команду в командный порт. Все что нужно это записать битовую маску в порт данных. В этой битовой маске входам, которые нужно замаскировать должна соответствовать 1, а тем, которые нужно разрешить должен соответствовать 0.

[1] Более точно для 80x86, в те времена этот контроллер прерываний и появился.

[2] Я в данном случае намеренно опускаю вопрос приоритезации и упрощаю рассказ.

[3] И нас устраивает способ, которым контроллер прерываний это делает.

Пока вы не знаете к какимм входам какое устройство подключено и не настроили соответсвующее устройство, вход, к которому оно подключено, лучше замаскировать. Т. е. полезно сразу после настройки контроллера прерываний замаскировать все прерывания и включать по мере настройки устройств.

3.6 Вывод

После прочтения этой части у вас в голове должна быть уже полная картина, т. е. настройка процессора и настройка контроллера прерываний, как маскировать прерывания на процессоре и как маскировать отдельные входы контоллера прерываний, как входы контроллера прерываний отображаются на записи в IDT, как, когда и зачем нужно посылать EOI контроллеру прерываний.