**Прерывания, исключения, системные вызовы**

Из теории ОС известно , что современные ОС реализуют поддержку системных вызовов, обработку прерываний и исключительных ситуаций, которые относят к основным механизмам ОС.

*Системные вызовы (system calls)* - механизм, позволяющий пользовательским программам обращаться к услугам ядра ОС, то есть это *интерфейс* между операционной системой и пользовательской программой. Концептуально *системный вызов* похож на обычный вызов подпрограммы. Основное отличие состоит в том, что при системном вызове выполнение программы осуществляется в привилегированном режиме или режиме ядра. Поэтому системные вызовы иногда еще называют *программными прерываниями*, в отличие от аппаратных прерываний, которые чаще называют просто прерываниями. В большинстве операционных систем *системный вызов* является результатом выполнения команды *программного прерывания* ( INT ). Таким образом, *системный вызов* - это *синхронное* событие.

*Прерывание (hardware interrupt)* - это событие, генерируемое внешним (по отношению к процессору) устройством. Посредством аппаратных прерываний аппаратура либо информирует центральный *процессор* о том, что произошло событие, требующее немедленной реакции (например, *пользователь* нажал клавишу), либо сообщает о завершении *операции* ввода вывода (например, закончено *чтение данных* с диска в основную *память*). Каждый тип аппаратных прерываний имеет собственный номер, однозначно определяющий источник прерывания. *Аппаратное прерывание* - это *асинхронное* событие, то есть оно возникает вне зависимости от того, какой код исполняется процессором в данный момент. Обработка аппаратного прерывания не должна учитывать, какой процесс или *поток* является текущим.

*Исключительная ситуация (exception)* - событие, возникающее в результате попытки выполнения программой команды, которая по каким-то причинам не может быть выполнена до конца. Примерами таких команд могут быть попытки доступа к ресурсу при отсутствии достаточных привилегий или обращение к отсутствующей странице памяти. Исключительные ситуации, как и системные вызовы, являются синхронными событиями, возникающими в контексте текущей задачи. Исключительные ситуации можно разделить на исправимые и неисправимые. К исправимым относятся такие исключительные ситуации, как отсутствие нужной информации в оперативной памяти. После устранения причины исправимой исключительной ситуации *программа* может выполняться дальше. Возникновение в процессе работы операционной системы исправимых исключительных ситуаций считается нормальным явлением. Неисправимые исключительные ситуации чаще всего возникают в результате ошибок в программах (например, *деление* на ноль). Обычно в таких случаях *операционная система* реагирует завершением программы, вызвавшей исключительную ситуацию.

Применяя структурную обработку исключений, упомянутую в предыдущей лекции, можно попытаться "исправить" неисправимую исключительную ситуацию, вернув управление программе, которая сгенерировала эту ситуацию.

**Реализация прерываний, системных вызовов и исключений в ОС Windows**

Рассмотрим реализацию основных механизмов операционной системы в ОС *Windows*. Следует отметить, что терминология корпорации Microsoft несколько отличается от общепринятой. Например, системные вызовы называются *системными сервисами*, а под программным прерыванием (см. прерывания *DPC* и *APC*) понимается выполнение специфичных функций ядра, требующих прерывания работы текущего процесса.

**Ловушки**

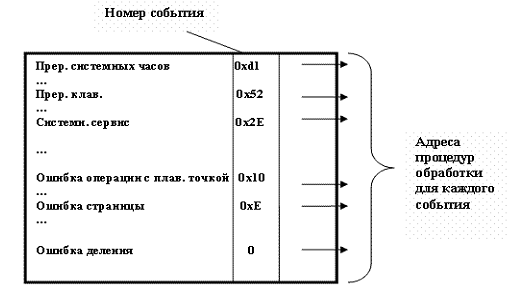
Общим для реализации рассматриваемых основных механизмов является *необходимость сохранения состояния текущего потока с его последующим восстановлением*. Для этого в ОС Windows используется механизм ловушек (trap). В случае возникновения требующего обработки события (прерывания, исключения или вызова *системного сервиса*) процессор переходит в привилегированный режим и передает управление обработчику ловушек, входящему в состав ядра. Обработчик ловушек создает в стеке ядра (о стеке ядра см. ["Реализация процессов и потоков"](https://www.intuit.ru/studies/courses/1089/217/lecture/5593)) прерываемого потока фрейм ловушки, содержащий часть контекста потока для последующего восстановления его состояния, и в свою очередь передает управление определенной части ОС, отвечающей за первичную обработку произошедшего события.

В типичном случае сохраняются и впоследствии восстанавливаются:

* программный счетчик;
* регистр состояния процессора;
* содержимое остальных регистров процессора;
* указатели на стек ядра и пользовательский стек;
* указатели на адресное пространство, в котором выполняется поток (каталог таблиц страниц процесса).

Эта информация специфицирована в структуре CONTEXT (файл winnt.h), и может быть получена пользователем с помощью функции GetThreadContext.

Адрес части ядра ОС, ответственной за обработку данного конкретного события определяется из *вектора прерываний*, который номеру события ставит в соответствие адрес процедуры его первичной обработки. Это оказывается возможным, поскольку все события типизированы и их число ограничено. Для асинхронных событий их номер определяется *контроллером прерываний*, а для синхронных - ядром. В [[ Руссинович ]](https://www.intuit.ru/studies/courses/1089/217/literature#literature.%D0%A0%D1%83%D1%81%D1%81%D0%B8%D0%BD%D0%BE%D0%B2%D0%B8%D1%87)описана процедура просмотра вектора прерываний, который в терминологии корпорации Microsoft называется таблицей диспетчеризации прерываний (*interrupt dispatch* table, *IDT*), при помощи отладчика kd. Например, для x86 процессора прерыванию от клавиатуры соответствует номер 0x52, системным сервисам - 0x2e, а исключительной ситуации, связанной со страничной ошибкой, - 0xE (см. [рис. 3.1](https://www.intuit.ru/studies/courses/1089/217/lecture/5589?page=1#image.3.1)рс. 3.1).



**Рис. 3.1.**Вектор прерываний (IDT)

После прохождения первичной обработки для каждого события предусмотрена процедура его последующей обработки другими частями ОС. Например, обработка *системного сервиса* (системного вызова) предполагает передачу управления по адресу 0x2e, где располагается диспетчер системных сервисов, которому через регистры EAX и EBX передаются номер запрошенного сервиса и список параметров, передаваемых этому системному сервису.

То же самое происходит в случае возникновения исключений и прерываний. Простые исключения могут быть обработаны диспетчером ловушек, а более сложные обрабатываются диспетчером исключений, который может в случае возникновения исключения вернуть управление вызвавшему это исключение приложению. Это делается с помощью упомянутого выше аппарата структурной обработки исключений. Вторичная обработка прерывания обеспечивается драйверами соответствующих устройств.

В качестве примера рассмотрим процедуру обработки создания файла. Вызов Win32 функции CreateFile() генерирует передачу управления функции NtCreateFile исполнительной системы, ассемблерный код которой содержит следующие операции:

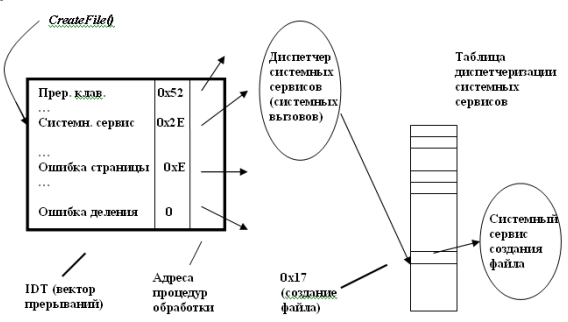
mov еах, Ox17 номер системного сервиса для NtCreateFile

mov ebx, esp

int Ox2E обработка системного сервиса

ret Ox2C возврат управления

[Рисунок 3.2](https://www.intuit.ru/studies/courses/1089/217/lecture/5589?page=1#image.3.2) иллюстрирует дальнейшую обработку данного сервиса.



**Рис. 3.2.**Пример обработки системного вызова (системного сервиса).

**Приоритеты. IRQL**

В большинстве операционных систем аппаратные прерывания имеют приоритеты, которые определяются контроллерами прерываний. Однако ОС Windows имеет свою аппаратно-независимую шкалу приоритетов, которые называются *уровни запросов прерываний* (*interrupt request* levels, IRQL), и охватывает не только прерывания, а все события, требующие системной обработки. В [таблице 3.1](https://www.intuit.ru/studies/courses/1089/217/lecture/5589?page=1#table.3.1) приведены значения IRQL уровней для x86 систем.

|  |  |  |
| --- | --- | --- |
| Таблица 3.1. Уровни запросов прерываний (IRQL) в x86 системах | | |
| **Уровень** | **Значение** | **Номер** |
| High | Наивысший уровень | 31 |
| Power fail | Отказ электропитания | 30 |
| Inter-process interrupt | *Межпроцессорный* сигнал | 29 |
| Clock | Системные часы | 28 |
| Profile | Контроль производительности ядра | 27 |
| Device n | Прерывание от устройства | 26 |
| … | Прерывания от устройств | … |
| Device 1 | Прерывание от устройства | 3 |
| *DPC*/dispatch | *Отложенные операции* и планирование | 2 |
| *APC* | Асинхронные вызовы процедур | 1 |
| *Passive* | Нормальное выполнение потоков | 0 |

Обрабатываемые события обслуживаются в порядке их приоритета, и события с более высоким приоритетом вытесняют обработку событий с меньшим приоритетом. При возникновении события с высоким приоритетом IRQL процессора повышается до уровня данного события. После его обработки могут проявить себя замаскированные менее приоритетные события, которые, в свою очередь, могут быть обработаны по обычной схеме. Текущий уровень приоритета хранится в данных, описывающих состояние процессора, и может быть определен системным отладчиком kd или посредством вызова функции KeGetCurrentIrql.

Значения IRQL для аппаратных прерываний расставляются диспетчером Plug and Play с помощью уровня абстрагирования от оборудования *HAL*, а для остальных событий - ядром. Таким образом, уровень IRQL определяется *источником события*, что имеет иной смысл, нежели приоритеты в стратегии планирования потоков. Разбиение на IRQL уровни является основным механизмом упорядочивания по приоритетам действий операционной системы.

Можно сказать, что в ОС Windows действует двухуровневая схема планирования. Приоритеты высшего уровня (в данном случае IRQLs) определяются аппаратными или *программными прерываниями*, а приоритеты низшего уровня (в своем диапазоне от 0 до 31) устанавливаются для пользовательских потоков, выполняемых на нулевом уровне IRQL, и контролируются планировщиком.

На нулевом (*PASSIVE* LEVEL) уровне IRQL работают пользовательские процессы и часть кода операционной системы. Программа, работающая на этом уровне, может быть вытеснена почти любым событием, случившимся в системе. Большинство процедур режима ядра старается удерживать IRQL уровень процессора как можно более низким.

IRQL уровни 1 (*APC* LEVEL) и 2 (DISPATCH LEVEL) предназначены для так называемых программных (в терминологии Microsoft) прерываний соответственно: асинхронный вызов процедуры - *APC* (asynchronous procedure call) и отложенный вызов процедуры - *DPC* (deferred procedure call). Если ядро принимает решение выполнить некоторую системную процедуру, но нет необходимости делать это немедленно, оно ставит ее в очередь *DPC* и генерирует *DPC* прерывание. Когда IRQL процессора станет достаточно низким, эта процедура выполняется. Характерный пример - отложенная операция планирования. Из этого следует, что код, выполняемый на IRQL уровне, выше или равном 2, не подвержен операции планирования. Асинхронный вызов процедур - механизм, аналогичный механизму *DPC*, но более общего назначения, в частности, доступный пользовательским процессам.

IRQL уровни 3-26 относятся к обычным прерываниям от устройств. Более подробно смотреть в литературе.

**Заключение**

В настоящей лекции описаны прерывания, системные вызовы и исключительные ситуации, которые являются фундаментальными механизмами операционных систем, и проанализированы особенности их реализации в ОС *Windows*. Обработка всех типов событий осуществляется единым образом и связана с сохранением/восстановлением состояния и эффективным поиском программы обработчика по системным таблицам. Важную роль для правильной организации имеет *иерархия* событий, реализованная в виде набора IRQL приоритетов.