**Многопото́чность** — свойство платформы (например, [операционной системы](https://ru.wikipedia.org/wiki/Операционная_система), [виртуальной машины](https://ru.wikipedia.org/wiki/Виртуальная_машина) и т. д.) или [приложения](https://ru.wikipedia.org/wiki/Прикладное_программное_обеспечение), состоящее в том, что[процесс](https://ru.wikipedia.org/wiki/Процесс_(информатика)), порождённый в операционной системе, может состоять из нескольких [*потоков*](https://ru.wikipedia.org/wiki/Поток_выполнения), выполняющихся «[параллельно](https://ru.wikipedia.org/wiki/Параллельные_вычисления)», то есть без предписанного порядка во [времени](https://ru.wikipedia.org/wiki/Время). При выполнении некоторых задач такое разделение может достичь более эффективного использования [ресурсов вычислительной машины](https://ru.wikipedia.org/wiki/Вычислительные_ресурсы).

Такие *потоки* называют также *потоками выполнения* (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) *[thread of execution](https://en.wikipedia.org/wiki/thread_(computer_science))*); иногда называют «нитями» (буквальный перевод [англ.](https://ru.wikipedia.org/wiki/Английский_язык) *thread*) или неформально «тредами».

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в [адресном пространстве](https://ru.wikipedia.org/wiki/Адресное_пространство) процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие [дескрипторы файлов](https://ru.wikipedia.org/wiki/Файловый_дескриптор). Выполняющийся процесс имеет как минимум один (главный) поток.

К достоинствам многопоточности в программировании можно отнести следующее:

* Упрощение программы в некоторых случаях за счет использования общего адресного пространства.
* Меньшие относительно процесса временны́е затраты на создание потока.
* Повышение производительности процесса за счет распараллеливания процессорных вычислений и операций ввода-вывода.
* Поток в пространстве пользователя. Каждый процесс имеет таблицу потоков, аналогичную таблице процессов ядра.

Достоинства и недостатки этого типа следующие: Недостатки

1. Отсутствие прерывания по таймеру внутри одного процесса
2. При использовании блокирующего системного запроса для процесса все его потоки блокируются.
3. Сложность реализации

* Поток в пространстве ядра. Наряду с таблицей процессов в пространстве ядра имеется таблица потоков.
* «Волокна» ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *fibers*). Несколько потоков режима пользователя, исполняющихся в одном потоке режима ядра. Поток пространства ядра потребляет заметные ресурсы, в первую очередь физическую память и диапазон адресов режима ядра для стека режима ядра. Поэтому было введено понятие «волокна» — облегчённого потока, выполняемого исключительно в режиме пользователя. У каждого потока может быть несколько «волокон».

Взаимодействие потоков

В многопоточной среде часто возникают проблемы, связанные с использованием параллельно исполняемыми потоками одних и тех же данных или устройств. Для решения подобных проблем используются такие методы взаимодействия потоков, как взаимоисключения (мьютексы), семафоры, критические секции и события

* Взаимоисключения (mutex, [мьютекс](https://ru.wikipedia.org/wiki/Мьютекс)) — это объект синхронизации, который устанавливается в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов (от английского **mut**ually **ex**clusive access — взаимно исключающий доступ) — одновременный доступ к общему ресурсу исключается. После всех необходимых действий мьютекс освобождается, предоставляя другим потокам доступ к общему ресурсу. Объект может поддерживать рекурсивный захват второй раз тем же потоком, увеличивая счетчик, не блокируя поток, и требуя потом многократного освобождения. Такова, например, критическая секция в [Win32](https://ru.wikipedia.org/wiki/Win32). Тем не менее есть и такие реализации, которые не поддерживают такое и приводят к [взаимной блокировке](https://ru.wikipedia.org/wiki/Взаимная_блокировка) потока при попытке рекурсивного захвата. Это FAST\_MUTEX в ядре Windows.
* [Семафоры](https://ru.wikipedia.org/wiki/Семафор_(информатика)) представляют собой доступные ресурсы, которые могут быть приобретены несколькими потоками в одно и то же время, пока пул ресурсов не опустеет. Тогда дополнительные потоки должны ждать, пока требуемое количество ресурсов не будет снова доступно. Семафоры очень эффективны, поскольку они позволяют одновременный доступ к ресурсам. Семафор есть логическое расширение мьютекса — семафор со счетчиком 1 эквивалентен мьютексу, но счетчик может быть и более 1.
* События. Объект, хранящий в себе 1 бит информации «просигнализирован или нет», над которым определены операции «просигнализировать», «сбросить в непросигнализированное состояние» и «ожидать». Ожидание на просигнализированном событии есть отсутствие операции с немедленным продолжением исполнения потока. Ожидание на непросигнализированном событии приводит к приостановке исполнения потока до тех пор, пока другой поток (или же вторая фаза [обработчика прерывания](https://ru.wikipedia.org/wiki/Обработчик_прерывания) в ядре ОС) не просигнализирует событие. Возможно ожидание нескольких событий в режимах «любого» или «всех». Возможно также создания события, автоматически сбрасываемого в непросигнализированное состояние после пробуждения первого же — и единственного — ожидающего потока (такой объект используется как основа для реализации объекта «критическая секция»). Активно используются в MS Windows, как в режиме пользователя, так и в режиме ядра. Аналогичный объект имеется и в ядре Linux под названием kwait\_queue.
* [Критические секции](https://ru.wikipedia.org/wiki/Критическая_секция) обеспечивают синхронизацию подобно мьютексам за исключением того, что объекты, представляющие критические секции, доступны в пределах одного процесса. События, мьютексы и семафоры также можно использовать в однопроцессном приложении, однако реализации критических секций в некоторых ОС (например, Windows NT) обеспечивают более быстрый и более эффективный[[1]](https://ru.wikipedia.org/wiki/Многопоточность#cite_note-1)[[2]](https://ru.wikipedia.org/wiki/Многопоточность#cite_note-2) механизм взаимно-исключающей синхронизации — операции «получить» и «освободить» на критической секции оптимизированы для случая единственного потока (отсутствия конкуренции) с целью избежать любых ведущих в ядро ОС системных вызовов. Подобно мьютексам объект, представляющий критическую секцию, может использоваться только одним потоком в данный момент времени, что делает их крайне полезными при разграничении доступа к общим ресурсам.
* [Условные переменные](https://ru.wikipedia.org/wiki/Условная_переменная) (condvars). Сходны с событиями, но не являются объектами, занимающими память — используется только адрес переменной, понятие «содержимое переменной» не существует, в качестве условной переменной может использоваться адрес произвольного объекта. В отличие от событий, установка условной переменной в просигнализированное состояние не влечет за собой никаких последствий в случае, если на данный момент нет потоков, ожидающих на переменной. Установка события в аналогичном случае влечет за собой запоминание состояния «просигнализировано» внутри самого события, после чего следующие потоки, желающие ожидать события, продолжают исполнение немедленно без остановки. Для полноценного использования такого объекта необходима также операция «освободить mutex и ожидать условную переменную атомарно». Активно используются в [UNIX](https://ru.wikipedia.org/wiki/UNIX)-подобных ОС. Дискуссии о преимуществах и недостатках событий и условных переменных являются заметной частью дискуссий о преимуществах и недостатках Windows и UNIX.
* Порт завершения ввода-вывода (IO completion port, IOCP). Реализованный в ядре ОС и доступный через системные вызовы объект «очередь» с операциями «поместить структуру в хвост очереди» и «взять следующую структуру с головы очереди» — последний вызов приостанавливает исполнение потока в случае, если очередь пуста, и до тех пор, пока другой поток не осуществит вызов «поместить». Самой важной особенностью IOCP является то, что структуры в него могут помещаться не только явным системным вызовом из режима пользователя, но и неявно внутри ядра ОС как результат завершения асинхронной операции ввода-вывода на одном из дескрипторов файлов. Для достижения такого эффекта необходимо использовать системный вызов «связать дескриптор файла с IOCP». В этом случае помещенная в очередь структура содержит в себе код ошибки операции ввода-вывода, а также, для случая успеха этой операции — число реально введенных или выведенных байт. Реализация порта завершения также ограничивает число потоков, исполняющихся на одном процессоре/ядре после получения структуры из очереди. Объект специфичен для MS Windows, и позволяет обработку входящих запросов соединения и порций данных в серверном программном обеспечении в архитектуре, где число потоков может быть меньше числа клиентов (нет требования создавать отдельный поток с расходами ресурсов на него для каждого нового клиента).
* ERESOURCE. Мьютекс, поддерживающий рекурсивный захват, с семантикой разделяемого или эксклюзивного захвата. Семантика: объект может быть либо свободен, либо захвачен произвольным числом потоков разделяемым образом, либо захвачен всего одним потоком эксклюзивным образом. Любые попытки осуществить захваты, нарушающее это правило, приводят к блокировке потока до тех пор, пока объект не освободится так, чтобы сделать захват разрешенным. Также есть операции вида TryToAcquire — никогда не блокирует поток, либо захватывает, либо (если нужна блокировка) возвращает FALSE, ничего не делая. Используется в ядре Windows, особенно в файловых системах — так, например, любому кем-то открытому дисковому файлу соответствует структура FCB, в которой есть 2 таких объекта для синхронизации доступа к размеру файла. Один из них — paging IO resource — захватывается эксклюзивно только в пути обрезания файла, и гарантирует, что в момент обрезания на файле нет активного ввода-вывода от кэша и от отображения в память.
* [Rundown protection](https://ru.wikipedia.org/wiki/Rundown_protection). Полудокументированный (вызовы присутствуют в файлах-заголовках, но отсутствуют в документации) объект в ядре Windows. Счетчик с операциями «увеличить», «уменьшить» и «ждать». Ожидание блокирует поток до тех пор, пока операции уменьшения не уменьшат счетчик до нуля. Кроме того, операция увеличения может отказать, и наличие активного в данный момент времени ожидания заставляет отказывать все операции увеличения.

**Создание процессов и потоков**

Создать процесс — это прежде всего означает создать описатель процесса, в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе,, необходимые операционной системе для управления им. В число таких сведений могут входить, например, идентификатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности процесса (приоритет и права доступа) и т. п. Примерами описателей процесса являются блок управления задачей (ТСВ — Task Control Block) в OS/360, управляющий блок процесса (РСВ — Process Control Block) в OS/2, дескриптор процесса в UNIX, объект-процесс (object-process) в Windows NT.

Создание описателя процесса знаменует собой появление в системе еще одного претендента на вычислительные ресурсы. Начиная с этого момента при распределении ресурсов ОС должна принимать во внимание потребности нового процесса.

Создание процесса включает загрузку кодов и данных исполняемой программы данного процесса с диска в оперативную память. Для этого ОС должна обнаружить местоположение такой программы на диске, перераспределить оперативную память и выделить память исполняемой программе нового процесса. Затем необходимо считать программу в выделенные для нее участки памяти и, возможно, изменить параметры программы в зависимости от размещения в памяти. В системах с виртуальной памятью в начальный момент может загружаться только часть кодов и данных процесса, с тем чтобы «подкачивать» остальные по мере необходимости. Существуют системы, в которых на этапе создания процесса не требуется непременно загружать коды и данные в оперативную память, вместо этого исполняемый модуль копируется из того каталога файловой системы, в котором он изначально находился, в область подкачки — специальную область диска, отведенную для хранения кодов и данных процессов. При выполнении всех этих действий подсистема управления процессами тесно взаимодействует с подсистемой управления памятью и файловой системой.

В многопоточной системе при создании процесса ОС создает для каждого процесса как минимум один поток выполнения. При создании потока так же, как и при создании процесса, операционная система генерирует специальную информационную структуру — описатель потока, который содержит идентификатор потока, данные о правах доступа и приоритете, о состоянии потока и другую информацию. В исходном состоянии поток (или процесс, если речь идет о системе, в которой понятие «поток» не определяется) находится в приостановленном состоянии. Момент выборки потока на выполнение осуществляется в соответствии с принятым в данной системе правилом предоставления процессорного времени и с учетом всех существующих в данный момент потоков и процессов. В случае если коды и данные процесса находятся в области подкачки, необходимым условием активизации потока процесса является также наличие места в оперативной памяти для загрузки его исполняемого модуля.

Во многих системах поток может обратиться к ОС с запросом на создание так называемых потоков-потомков. В разных ОС по-разному строятся отношения между потоками-потомками и их родителями. Например, в одних ОС выполнение родительского потока синхронизируется с его потомками, в частности после завершения родительского потока ОС может снимать с выполнения всех его потомков. В других системах потоки-потомки могут выполняться асинхронно по отношению к родительскому потоку. Потомки, как правило, наследуют многие свойства родительских потоков. Во многих системах порождение потомков является основным механизмом создания процессов и потоков.

Рассмотрим в качестве примера создание процессов в популярной версии операционной системы UNIX System V Release 4. В этой системе потоки не поддерживаются, в качестве единицы управления и единицы потребления ресурсов выступает процесс.

При управлении процессами операционная система использует два основных типа информационных структур: дескриптор процесса и контекст процесса. Дескриптор процесса содержит такую информацию о процессе, которая необходима ядру в течение всего жизненного цикла процесса независимо от того, находится он в активном или пассивном состоянии, находится образ процесса в оперативной памяти или выгружен на диск. (Образом процесса называется совокупность его кодов и данных.)

Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов. В дескрипторе прямо или косвенно (через указатели, на связанные с процессом структуры) содержится информация о состоянии процесса, о расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, а также о его итоговом значении — глобальном приоритете, об идентификаторе пользователя, создавшего процесс, о родственных процессах, о событиях, осуществления которых ожидает данный процесс, и некоторая другая информация.

Контекст процесса содержит менее оперативную, но более объемную часть информации о процессе, необходимую для возобновления выполнения процесса с прерванного места: содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, информация обо всех открытых данным процессом файлах и незавершенных операциях ввода-вывода и другие данные, характеризующие состояние вычислительной среды в момент прерывания. Контекст, так же как и дескриптор процесса, доступен только программам ядра, то есть находится в виртуальном адресном пространстве операционной системы, однако он хранится не в области ядра, а непосредственно примыкает к образу процесса и перемещается вместе с ним, если это необходимо, из оперативной памяти на диск.

Порождение процессов в системе UNIX происходит в результате выполнения системного вызова fork. ОС строит образ порожденного процесса являющийся точной копией образа породившего процесса, то есть дублируются дескриптор, контекст и образ процесса. Сегмент данных и сегмент стека родительского процесса копируются на новое место, образуя сегменты данных и стека процесса-потомка. Процедурный сегмент копируется только тогда, когда он не является разделяемым. В противном случае процесс-потомок становится еще одним процессом, разделяющим данный процедурный сегмент.

После выполнения системного вызова fork оба процесса продолжают выполнение с одной и той же точки. Чтобы процесс мог опознать, является он родительским процессом или процессом-потомком, системный вызов fork возвращает в качестве своего значения в породивший процесс идентификатор порожденного процесса, а в порожденный процесс — NULL. Типичное разветвление на языке С записывается так:

if( -fork() ) { действия родительского процесса }

else { действия порожденного процесса }

Идентификатор потомка может быть присвоен переменной, входящей в контекст родительского процесса. Так как контекст процесса наследуется его потомками, то потомки могут узнать идентификаторы своих «старших братьев», таким образом сумма знаний наследуется при порождении и может быть распространена между родственными процессами. На независимости идентификатора процесса от выполняемой процессом программы построен механизм, позволяющий процессу перейти к выполнению другой программы с помощью системного вызова ехес.

Таким образом, в UNIX порождение нового процесса происходит в два этапа — сначала создается копия процесса-родителя, затем у нового процесса производится замена кодового сегмента на заданный.

Вновь созданному процессу операционная система присваивает целочисленный идентификатор, уникальный на весь период функционирования системы.

Состояния потока

ОС выполняет планирование потоков, принимая во внимание их состояние. В мультипрограммной системе поток может находиться в одном из трех основных состояний:

* выполнение — активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
* ожидание — пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например завершения операции ввода-вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса);
* готовность — также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

**Механизм прерываний**

Механизм прерываний поддерживается аппаратными средствами компьютера и программными средствами операционной системы. Аппаратная поддержка прерываний имеет свои особенности, зависящие от типа процессора и других аппаратных компонентов, передающих сигнал запроса прерывания от внешнего устройства к процессору (таких, как контроллер внешнего устройства, шины подключения внешних устройств, контроллер прерываний, являющийся посредником между сигналами шины и сигналами процессора). Особенности аппаратной peaлизации прерываний оказывают влияние на средства программной поддержки прерываний, работающие в составе ОС.

Существуют два основных способа, с помощью которых шины выполняют прерывания: векторный (vectored) и опрашиваемый (polled). В обоих способах процессору предоставляется информация об уровне приоритета прерывания на шине подключения внешних устройств. В случае векторных прерываний в процессор передается также информация о начальном адресе программы обработки возникшего прерывания — обработчика прерываний.

Устройствам, которые используют векторные прерывания, назначается вектор прерываний. Он представляет собой электрический сигнал, выставляемый на соответствующие шины процессора и несущий в себе информацию об определенном, закрепленном за данным устройством номере, который идентифицирует соответствующий обработчик прерываний. Этот вектор может быть фиксированным, конфигурируемым (например, с использованием переключателей) или программируемым. Операционная система может предусматривать процедуру регистрации вектора обработки прерываний для определенного устройства, которая связывает некоторую подпрограмму обработки прерываний с определенным вектором. При получении сигнала запроса прерывания процессор выполняет специальный цикл подтверждения прерывания, в котором устройство должно идентифицировать себя. В течение этого цикла устройство отвечает, выставляя на шину вектор прерываний. Затем процессор использует этот вектор для нахождения обработчика данного прерывания. Примером шины подключения внешних устройств, которая поддерживает векторные прерывания, является шина VMEbus.

При использовании опрашиваемых прерываний процессор получает от запросившего прерывание устройства только информацию об уровне приоритета прерывания (например, номере IRQ на шине ISA или номере IPL на шине SBus компьютеров SPARC). С каждым уровнем прерываний может быть связано несколько устройств и соответственно несколько программ — обработчиков прерываний. При возникновении прерывания процессор должен определить, какое устройство из тех, которые связаны с данным уровнем прерываний, действительно запросило прерывание. Это достигается вызовом всех обработчиков прерываний для данного уровня приоритета, пока один из обработчиков не подтвердит, что прерывание пришло от обслуживаемого им устройства. Если же с каждым уровнем прерываний связано только одно устройство, то определение нужной программы обработки прерывания происходит немедленно, как и при векторном прерывании. Опрашиваемые прерывания поддерживают шины ISA, EISA, MCA, PCI и Sbus.

Механизм прерываний некоторой аппаратной платформы может сочетать векторный и опрашиваемый типы прерываний. Типичным примером такой реализации является платформа персональных компьютеров на основе процессоров Intel Pentium. Шины PCI, ISA, EISA или MCA, используемые в этой платформе в качестве шин подключения внешних устройств, поддерживают механизм опрашиваемых прерываний. Контроллеры периферийных устройств выставляют на шину не вектор, а сигнал запроса прерывания определенного уровня IRQ. Однако в процессоре Pentium система прерываний является векторной. Вектор прерываний в процессор Pentium поставляет контроллер прерываний, который отображает поступающий от шины сигнал IRQ на определенный номер вектора.

Вектор прерываний, передаваемый в процессор, представляет собой целое число в диапазоне от 0 до 255, указывающее на одну из 256 программ обработки прерываний, адреса которых хранятся в таблице обработчиков прерываний. В том случае, когда к каждой линии IRQ подключается только одно устройство, процедура обработки прерываний работает так, как если бы система прерываний была чисто векторной, то есть процедура не выполняет никаких дополнительных опросов для выяснения того, какое именно устройство запросило прерывание. Однако при совместном использовании одного уровня IRQ несколькими устройствами программа обработки прерываний должна работать в соответствии со схемой опрашиваемых прерываний, то есть дополнительно выполнить опрос всех устройств, подключенных к данному уровню IRQ.

Механизм прерываний чаще всего поддерживает приоритезацию и маскирование прерываний. Приоритезация означает, что все источники прерываний делятся на классы и каждому классу назначается свой уровень приоритета запроса на прерывание. Приоритеты могут обслуживаться как относительные и абсолютные. Обслуживание запросов прерываний по схеме с относительными приоритетами заключается в том, что при одновременном поступлении запросов прерываний из разных классов выбирается запрос, имеющий высший приоритет. Однако в дальнейшем при обслуживании этого запроса процедура обработки прерывания уже не откладывается даже в том случае, когда появляются более приоритетные запросы — решение о выборе нового запроса принимается только в момент завершения обслуживания очередного прерывания. Если же более приоритетным прерываниям разрешается приостанавливать работу процедур обслуживания менее приоритетных прерываний, то это означает, что работает схема приоритезации с абсолютными приоритетами.

Если процессор (или компьютер, когда поддержка приоритезации прерываний вынесена во внешний по отношению к процессору блок) работает по схеме с абсолютными приоритетами, то он поддерживает в одном из своих внутренних регистров переменную, фиксирующую уровень приоритета обслуживаемого в данный момент прерывания. При поступлении запроса из определенного класса его приоритет сравнивается с текущим приоритетом процессора, и если приоритет запроса выше, то текущая процедура обработки прерываний вытесняется, а по завершении обслуживания нового прерывания происходит возврат к прерванной процедуре.

Упорядоченное обслуживание запросов прерываний наряду со схемами приоритетной обработки запросов может выполняться механизмом маскирования запросов. Собственно говоря, в описанной схеме абсолютных приоритетов выполняется маскирование — при обслуживании некоторого запроса все запросы с равным или более низким приоритетом маскируются, то есть не обслуживаются. Схема маскирования предполагает возможность временного маскирования прерываний любого класса независимо от уровня приоритета.

Обобщенно последовательность действий аппаратных и программных средств по обработке прерывания можно описать следующим образом.

1. При возникновении сигнала (для аппаратных прерываний) или условия (для внутренних прерываний) прерывания происходит первичное аппаратное распознавание типа прерывания. Если прерывания данного типа в настоящий момент запрещены (приоритетной схемой или механизмом маскирования), то процессор продолжает поддерживать естественный ход выполнения команд. В противном случае в зависимости от поступившей в процессор информации (уровень прерывания, вектор прерывания или тип условия внутреннего прерывания) происходит автоматический вызов процедуры обработки прерывания, адрес которой находится в специальной таблице операционной системы, размещаемой либо в регистрах процессора, либо в определенном месте оперативной памяти.

2. Автоматически сохраняется некоторая часть контекста прерванного потока, которая позволит ядру возобновить исполнение потока процесса после обработки прерывания. В это подмножество обычно включаются значения счетчика команд, слова состояния машины, хранящего признаки основных режимов работы процессора (пример такого слова — регистр EFLA6S в Intel Pentium), а также нескольких регистров общего назначения, которые требуются программе обработки прерывания. Может быть сохранен и полный контекст процесса, если ОС обслуживает данное прерывание со сменой процесса. Однако в общем случае это не обязательно, часто обработка прерываний выполняется без вытеснения текущего процесса1.

1 Решение о перепланировании процессов может быть принято в ходе обработки прерывания, например, если это прерывание от таймера и после наращивания значения системных часов выясняется, что процесс исчерпал выделенный ему квант времени. Однако это совсем не обязательно — прерывание может выполняться и без смены процесса, например прием очередной порции данных от контроллера внешнего устройства чаще всего происходит в рамках текущего процесса, хотя данные, скорее всего, предназначены другому процессу.

3. Одновременно с загрузкой адреса процедуры обработки прерываний в счетчик команд может автоматически выполняться загрузка нового значения слова состояния машины (или другой системной структуры, например селектора кодового сегмента в процессоре Pentium), которое определяет режимы работы процессора при обработке прерывания, в том числе работу в привилегированном режиме. В некоторых моделях процессоров переход в привилегированный режим за счет смены состояния машины при обработке прерывания является единственным способом смены режима. Прерывания практически во всех мультипрограммных ОС обрабатываются в привилегированном режиме модулями ядра, так как при этом обычно нужно выполнить ряд критических операций, от которых зависит жизнеспособность системы, — управлять внешними устройствами, перепланировать потоки и т. п.

4. Временно запрещаются прерывания данного типа, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры. Детали выполнения этой операции зависят от особенностей аппаратной платформы, например может использоваться механизм маскирования прерываний. Многие процессоры автоматически устанавливают признак запрета прерываний в начале цикла обработки прерывания, в противном случае это делает программа обработки прерываний.

5. После того как прерывание обработано ядром операционной системы, прерванный контекст восстанавливается и работа потока возобновляется с прерванного места. Часть контекста восстанавливается аппаратно по команде возврата из прерываний (например, адрес следующей команды и слово состояния машины), а часть — программным способом, с помощью явных команд извлечения данных из стека. При возврате из прерывания блокировка повторных прерываний данного типа снимается.

Программные прерывания

Программное прерывание реализует один из способов перехода на подпрограмму с помощью специальной инструкции процессора, такой как INT в процессорах Intel Pentium, trap в процессорах Motorola, syscall в процессорах MIPS или Ticc в процессорах SPARC. При выполнении команды программного прерывания процессор отрабатывает ту же последовательность действий, что и при возникновении внешнего или внутреннего прерывания, но только происходит это в предсказуемой точке программы — там, где программист поместил данную команду.

Практически все современные процессоры имеют в системе команд инструкции программных прерываний. Одной из причин появления инструкций программных прерываний в системе команд процессоров является то, что их использование часто приводит к более компактному коду программ по сравнению с использованием стандартных команд выполнения процедур. Это объясняется тем, что разработчики процессора обычно резервируют для обработки прерываний небольшое число возможных подпрограмм, так что длина операнда в команде программного прерывания, который указывает на нужную подпрограмму, меньше, чем в команде перехода на подпрограмму. Например, в процессоре х86 предусмотрена возможность применения 256 программ обработки прерываний, поэтому в инструкции INT операнд имеет длину в один байт (а инструкция INT 3, которая предназначена для вызова отладчика, вся имеет длину один байт). Значение операнда команды INT просто является индексом в таблице из 256 адресов подпрограмм обработки прерываний, один из которых и используется для перехода по команде INT. При использовании команды CALL потребовался бы уже не однобайтовый, а двух- или четырехбайтовый операнд. Другой причиной применения программных прерываний вместо обычных инструкций вызова подпрограмм является возможность смены пользовательского режима на привилегированный одновременно с вызовом процедуры — это свойство программных прерываний поддерживается большинством процессоров.

В результате программные прерывания часто используются для выполнения ограниченного количества вызовов функций ядра операционной системы, то есть системных вызовов.