1. Классификация операционных систем и особенности их функционирования.

Различают следующие типы операционных систем:

- Однопрограммные ОС
- Многопрограммные ОС
- Однопроцессорные ОС
- Многопроцессорные ОС
- Распределенные ОС
- Виртуальные ОС

Система работает в ***однопрограммном*** режиме если в системе находится одна или несколько задач, но все ресурсы системы отданы одной задаче, и она не может быть прервана другой. Задача возвращает ресурсы только при нормальном или аварийном завершении.

Система работает в ***многопрограммном*** режиме, если в ней находится несколько задач в разной стадии исполнения, и каждая из них может быть прервана другой с последующим возвратом. Разделение аппаратных и программных ресурсов системы ставит сложные задачи управления перед **СУПЕРВИЗОР**ом, которые он решает благодаря наличию специальных алгоритмов распределения рессурсов системы. Принято различать **пять способов реализации** многопрограммного режима работы:

- Классическое мультипрограммирование;
- Параллельная обработка;
- Разделение времени;
- Свопинг.

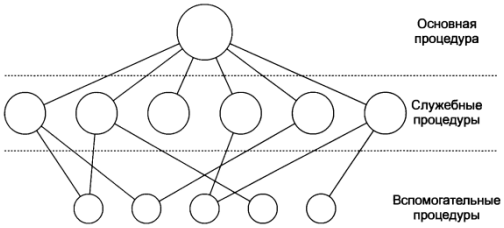
Многопрограммный режим в однопроцессорной системе — это организация вычислительного процесса с планированием во времени. При этом операционная система обеспечивает выполнение активизированных процессов на одном и том же оборудовании, разделяя (синхронизируя) их работу во времени. В функции такой операционной системы также входит защита влияния одного процесса на другой. Основные заботы о синхронизации взаимодействующих процессов возложены на программиста.

*Многопрограммный режим в **многопроцессорной** системе* — это планирование во времени и в пространстве. В этом режиме операционная система, выполняя одну из своих основных функций (обеспечение эффективности работы ресурсов), должна распределять активизированные процессы по процессорам (в пространстве) и синхронизировать их работу во времени. В том случае, если количество задач больше количества процессоров, задача планирования, диспетчеризации усложняется. В связи со сложностью решения задач распределения процессов по процессорам, их решение выполняется или до активизации процессов в многопроцессорной системе (статическое планирование), или решаются простыми способами, не дающими оптимального решения.

По структуре:

А) монолитные - Здесь вся операционная система работает как единая программа в режиме ядра. Операционная система написана в виде набора процедур, связанных вместе в одну большую исполняемую программу При использовании этой технологии каждая процедура может свободно вызвать любую другую процедуру, если та выполняет какое-нибудь полезное действие, в котором нуждается первая процедура. Наличие нескольких тысяч процедур, которые могут вызывать друг друга сколь угодно часто, нередко приводит к громоздкой и непонятной системе. Для построения исполняемого файла монолитной системы необходимо сначала скомпилировать все отдельные процедуры (или файлы, содержащие процедуры), а затем связать их все вместе, воспользовавшись системным компоновщиком. Здесь, по существу, полностью отсутствует сокрытие деталей реализации — каждая процедура видна любой другой процедуре (в отличие от структуры, содержащей модули или пакеты, в которых основная часть информации скрыта внутри модулей,
и за пределами модуля его процедуры можно вызвать только через специально определяемые точки входа). Тем не менее даже такие монолитные системы могут иметь некоторую структуру. Службы (системные вызовы), предоставляемые операционной системой, запрашиваются путем помещения параметров в четко определенное место (например, в стек), а затем выполняется инструкция trap. Эта инструкция переключает машину из пользовательского режима в режим ядра и передает управление операционной системе. Затем операционная система извлекает параметры и определяет, какой системный вызов должен быть выполнен. После этого она перемещается по индексу в таблице, которая в строке k содержит указатель на процедуру, выполняющую системный вызов k.
Такая организация предполагает следующую базовую структуру операционной системы:

1. Основная программа, которая вызывает требуемую служебную процедуру.
2. Набор служебных процедур, выполняющих системные вызовы.
3. Набор вспомогательных процедур, содействующих работе служебных процедур.



В дополнение к основной операционной системе, загружаемой во время запуска компьютера, многие операционные системы поддерживают загружаемые расширения, в числе которых драйверы устройств ввода-вывода и файловые системы. Эти компоненты загружаются по мере надобности.

Б) многоуровневые системы - Обобщением подход, является организация операционной системы в виде иерархии уровней, каждый из которых является надстройкой над нижележащим уровнем. у системы было шесть уровней. **Уровень 0** занимался распределением ресурса процессора (процессорного времени), переключением между процессами при возникновении прерываний или истечении времени таймера. Над уровнем 0 система состояла из последовательных процессов, каждый из которых мог быть запрограммирован без учета того, что несколько процессов были запущены на одном процессоре. Иными словами, уровень 0 обеспечивал основу многозадачности центрального процессора. **Уровень 1** управлял памятью. Он выделял процессам пространство в основной памяти и на магнитном барабане емкостью 512 К слов, который использовался для хранения частей процесса (страниц), не умещавшихся в оперативной памяти. На уровнях выше первого процессы не должны были беспокоиться о том, где именно они находятся, в памяти или на барабане; доставкой страниц в память по мере надобности занималось программное обеспечение уровня 1. **Уровень 2** управлял связью каждого процесса с консолью оператора (то есть с пользователем). Над этим уровнем каждый процесс фактически имел свою собственную консоль оператора. **Уровень 3** управлял устройствами ввода-вывода и буферизацией информационных потоков в обоих направлениях. Над третьим уровнем каждый процесс мог работать с абстрактными устройствами ввода-вывода, имеющими определенные свойства. **На уровне 4** работали пользовательские программы, которым не надо было заботиться о процессах, памяти, консоли или
управлении вводом-выводом. Процесс системного оператора размещался **на уровне 5**.

Уровень	Функция
5	Оператор
4	Программы пользователя
3	Управление вводом-выводом
2	Связь оператора с процессом
1	Управление основной памятью и магнитным барабаном
0	Распределение ресурсов процессора и обеспечение многозадачного режима

Дальнейшее обобщение многоуровневой концепции было сделано в системе MULTICS. Вместо уровней для описания MULTICS использовались серии концентрических колец, где внутренние кольца обладали более высокими привилегиями по отношению к внешним (что, собственно, не меняло сути многоуровневой системы). Когда процедуре из внешнего кольца требовалось вызвать процедуру внутреннего кольца, ей нужно было создать эквивалент системного вызова, то есть выполнить инструкцию TRAP, параметры которой тщательно проверялись на допустимость перед тем, как разрешить продолжение вызова. Хотя вся операционная система в MULTICS являлась частью адресного пространства каждого пользовательского процесса, аппаратура позволяла определять отдельные процедуры (а фактически сегменты памяти) как защищенные от чтения, записи или выполнения.

Преимущества кольцеобразного механизма проявлялись в том, что он мог быть легко расширен и на структуру пользовательских подсистем. Например, профессор может написать программу для тестирования и оценки студенческих программ и запустить эту программу в кольце п, а студенческие программы будут выполняться в кольце п + 1, так что студенты не смогут изменить свои оценки.

В) Микроядра - При использовании многоуровневого подхода разработчикам необходимо выбрать, где провести границу между режимами ядра и пользователя. Традиционно все уровни входили в ядро, но это необязательно. Существуют очень весомые аргументы в пользу того, чтобы в режиме ядра выполнялось как можно меньше процессов, поскольку ошибки в ядре могут вызвать немедленный сбой системы. Для сравнения пользовательские процессы могут быть настроены на обладание меньшими полномочиями, чтобы их ошибки не носили фатального характера. Замысел, положенный в основу конструкции микроядра, направлен на достижение высокой надежности за счет разбиения операционной системы на небольшие, вполне определенные модули. Только один из них — микроядро — запускается в режиме ядра, а все остальные запускаются в виде относительно слабо наделенных полномочиями обычных пользовательских процессов. В частности, если запустить
каждый драйвер устройства и файловую систему как отдельные пользовательские процессы, то ошибка в одном из них может вызвать отказ соответствующего компонента, но не сможет вызвать сбой всей системы. Таким образом, ошибка в драйвере звукового устройства приведет к искажению или пропаданию звука, но не вызовет зависание компьютера. В отличие от этого в монолитной системе, где все драйверы находятся в ядре, некорректный драйвер звукового устройства может запросто сослаться на неверный адрес памяти и привести систему к немедленной вынужденной остановке. За пределами ядра структура системы представляет собой три уровня процессов, которые работают в режиме пользователя. Самый нижний уровень содержит драйверы устройств. Поскольку они работают в пользовательском режиме, у них нет физического доступа к пространству портов ввода-вывода и они не могут вызывать команды ввода-вывода напрямую. Вместо этого, чтобы запрограммировать устройство ввода-вывода, драйвер создает структуру, сообщающую, какие значение в какие порты ввода-вывода следует записать. Затем драйвер осуществляет вызов ядра, сообщая ядру, что нужно произвести запись. При этом ядро может осуществить проверку, использует ли драйвер то устройство ввода-вывода, с

которым он имеет право работать. Следовательно (в отличие от монолитной конструкции), дефектный драйвер звукового устройства не может случайно осуществить запись на диск.

Над драйверами расположен уровень, содержащий службы, которые осуществляют основной объем работы операционной системы. Все они работают в режиме пользователя. Одна или более файловых служб управляют файловой системой (или системами), диспетчер процессов создает, уничтожает процессы, управляет ими и т. д. Пользовательские программы получают доступ к услугам операционной системы путем отправки коротких сообщений к этим службам, которые запрашивают системные вызовы POSIX. Например, процесс, нуждающийся в выполнении вызова read} отправляет сообщение одной из файловых служб, предписывая ей, что нужно прочитать.

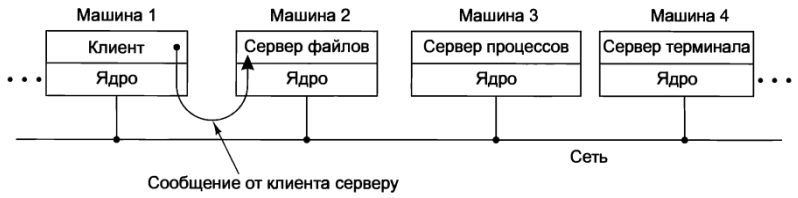
Отдельный интерес представляет служба перевоплощения (reincarnation server), выполняющая задачу по проверке функционирования других служб и драйверов. В случае обнаружения отказа одного из компонентов он автоматически заменяется без какого-либо вмешательства со стороны пользователя. Таким образом, система самостоятельно исправляет отказы и может достичь высокой надежности. Система накладывает на полномочия каждого процесса большое количество ограничений. Уже упоминалось, что драйверы могут работать только с разрешенными портами ввода-вывода. Доступ к вызовам ядра также контролируется для каждого процесса через возможность посылать сообщения другим процессам. Процессы также могут предоставить другим процессам ограниченные права доступа через ядро к своему адресному пространству. Например, файловая система может выдать разрешение драйверу диска, позволяющее ядру помещать только что считанный с диска блок в память по указанному адресу внутри адресного пространства файловой системы. Совокупность этих ограничений приводит к тому, что каждый драйвер и каждая служба имеют только те полномочия, которые нужны для их работы, и не более того. Тем самым существенно сокращается вред, который может быть нанесен дефектным компонентом.

Идея, имеющая некоторое отношение к использованию минимального ядра, заключается в том, чтобы помещать в ядро исполнительный механизм, а не политику. Чтобы пояснить эту мысль, рассмотрим планирование выполнения процессов. Относительно простой алгоритм планирования заключается в назначении каждому процессу приоритета с последующим запуском ядром готового к выполнению процесса с наиболее высоким приоритетом. Механизм, который находится в ядре, предназначен для поиска и запуска процесса с наибольшим приоритетом. Политика, заключающаяся в назначении процессам приоритетов, должна быть реализована процессами, работающими в пользовательском режиме. Таким образом, политика и механизм могут быть разобщены, а ядро может быть уменьшено в размерах.

Г) Клиент-сервер - Небольшая вариация идеи микроядер выражается в обособлении двух классов процессов: серверов, каждый из которых предоставляет какую-нибудь службу, и клиентов, которые пользуются этими службами. Эта модель известна как клиент- серверная. Достаточно часто самый нижний уровень представлен микроядром, но это не обязательно. Вся суть заключается в наличии клиентских процессов и серверных процессов.

Связь между клиентами и серверами часто организуется с помощью передачи сообщений. Чтобы воспользоваться службой, клиентский процесс составляет сообщение, в котором говорится, что именно ему нужно, и отправляет его соответствующей службе. Затем служба выполняет определенную работу и отправляет обратно ответ. Если клиент и сервер запущены на одной и той же машине, то можно провести определенную оптимизацию, но концептуально здесь идет речь о передаче сообщений.

Очевидным развитием этой идеи будет запуск клиентов и серверов на разных компьютерах, соединенных локальной или глобальной сетью, как показано на рис.. Поскольку клиенты связываются с серверами путем отправки сообщений, им не обязательно знать, будут ли эти сообщения обработаны локально, на их собственных машинах, или же они будут отправлены по сети на серверы, расположенные на удаленных машинах. Что касается интересов клиента, следует отметить, что в обоих случаях происходит одно и то же: отправляются запросы и возвращаются ответы. Таким образом, клиент-серверная модель является абстракцией, которая может быть использована как для отдельно взятой машины, так и для машин, объединенных в сеть.



Становится все больше и больше систем, привлекающих пользователей, сидящих за домашними компьютерами, в качестве клиентов, а большие машины, работающие где-нибудь в другом месте, — в качестве серверов. Фактически по этой схеме работает большая часть Интернета. Персональные компьютеры отправляют запросы на получение веб-страницы на сервер, и им возвращается эта веб-страница. Это типичная картина использования клиент-серверной модели при работе в сети.

Д) Виртуальные машины - Сначала о потребностях. Многие компании традиционно запускали свои

почтовые серверы, веб-серверы, FTP-серверы и все остальные серверы на отдельных компьютерах, иногда имеющих различные операционные системы. Виртуализация рассматривается ими как способ запуска всех этих серверов на одной и той же машине, избегая при этом отказа всех серверов при отказе одного из них. Виртуализация также популярна в мире веб-хостинга. Без нее клиенты вынуждены выбирать между общим хостингом1 (который дает им только учетную запись на веб-сервере, но не позволяет управлять его программным обеспечением) и выделенным хостингом (который предоставляет им их собственную, очень гибкую, но не оправдывающую затрат машину при небольших или средних по объему веб-сайтах). Когда компания, предоставляющая услуги веб-хостинга (хостинг- провайдер), сдает в аренду виртуальные машины, на одной физической машине может быть запущено множество виртуальных машин, каждая из которых превращается во вполне полноценную машину. Клиенты, арендовавшие виртуальную машину, могут запускать на ней какую угодно операционную систему и программное обеспечение, но за часть стоимости выделенного сервера (поскольку та же самая физическая машина одновременно поддерживает множество виртуальных машин).

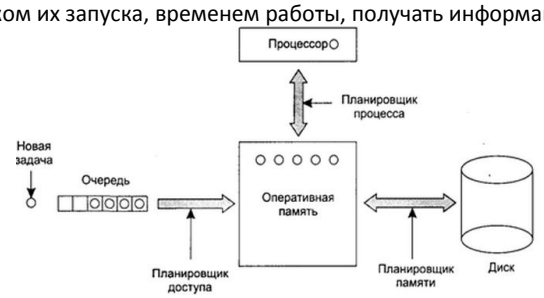
Другой вариант использования виртуализации предназначен для конечных пользователей, которым необходима возможность одновременного запуска двух или более операционных систем, например Windows и Linux, поскольку некоторые из любимых ими приложений работают под управлением только одной, а некоторые под управлением только

другой операционной системы. Такая ситуация показана на рис. 1.25, я, при этом термин «монитор виртуальной машины» заменен на «**гипервизор** первого типа» в соответствии с терминологией последних лет.
Теперь о программном обеспечении. Привлекательность виртуальных машин сомнениям не подвергалась, проблема заключалась в их реализации. Чтобы запустить на компьютере программное обеспечение виртуальных машин, его центральный процессор должен быть готов к работе в этом режиме. Проблема заключается в следующем. Когда операционная система, запущенная на виртуальной машине (в режиме пользователя) выполняет привилегированные инструкции, например изменение слова состояния программы — PSW или осуществление операции ввода-вывода, необходимо, чтобы оборудование осуществило ее перехват и вызов монитора виртуальных машин, который выполнит программную эмуляцию данной инструкции. На некоторых центральных процессорах, особенно на Pentium, его предшественниках и их клонах, попытки выполнения привилегированных инструкций в режиме пользователя просто игнорируются. Эта особенность исключает создание виртуальных машин на таком оборудовании, чем объясняется недостаточный интерес к ним в мире персональных компьютеров. Конечно, существовали интерпретаторы для Pentium, которые запускались на этом процессоре, но при потере производительности обычно в 5-10 раз они не подходили для серьезной работы.
Ж) Экзоядро - Вместо клонирования настоящей машины, как это делается в виртуальных машинах, существует иная стратегия, которая заключается в их разделении, иными словами, в предоставлении каждому пользователю подмножества ресурсов. При этом одна виртуальная машина может получить дисковые блоки от 0 до 1023, другая может получить блоки от 1024 до 2047, и т. д. Самый нижний уровень, работающий в режиме ядра, — это программа под названием экзоядро. Его задача состоит в распределении ресурсов между виртуальными машинами и затем в отслеживании попыток их использования, чтобы ни одна из машин не пыталась использовать чужие ресурсы. Каждая виртуальная машина может запускать свою собственную операционную систему, как на VM/370 и на Pentium в режиме виртуальных машин 8086, с тем отличием, что каждая машина ограничена использованием тех ресурсов, которые она запросила и которые были ей предоставлены. Преимущество схемы экзоядра заключается в том, что она исключает уровень отображения. При других методах работы каждая виртуальная машина считает, что она имеет свой собственный диск с нумерацией блоков от 0 до некоторого максимума. Поэтому монитор виртуальных машин должен вести таблицы преобразования адресов на диске (и всех других ресурсов). При использовании экзоядра необходимость в таком переназначении отпадает. Экзоядру нужно лишь отслеживать, какой виртуальной машине какие ресурсы были переданы. Такой подход имеет еще одно преимущество: он отделяет многозадачность (в экзоядре) от пользовательской операционной системы (в пространстве пользователя) с меньшими затратами, так как для этого ему необходимо всего лишь не допускать вмешательства одной виртуальной машины в работу другой.

2. Управление заданиями. Состав. Функции. Взаимодействие с другими частями ОС.

Задание – внешняя единица работы системы, на которую система ресурсы не выделяет. Работа, которую мы хотим дать ОС, чтоб она её выполнила. Задания накапливаются в буферах. В задании должна быть указана программа и данные. Как только появляются ресурсы, задание расшифровывается. Формируется task control block (TCB), он же process control block (PCB).
Задача – внутренняя единица системы, для которой система выделяет ресурсы (активизирует задание).
Процесс – любая выполняемая программа системы; это динамический объект системы, которому она выделяет ресурсы; траектория процессора в адресном пространстве машины. ОС всегда должна знать, что происходит с процессом.

Система управления заданиями предназначена для управления прохождением задач на многопроцессорных вычислительных установках (в том числе кластерных). Она позволяет автоматически распределять вычислительные ресурсы между задачами, управлять порядком их запуска, временем работы, получать информацию о состоянии очередей.



1 уровень. Задание кто-то формирует. Программист или система. В первом случае, если задание правильно оформлено (синтаксис), оно накапливается в очереди входных заданий.
Для выбора заданий из очереди может использоваться алгоритм, в котором устанавливается приоритет коротких задач перед длинными. Впускной/входной планировщик волен придержать некоторые задания во входной очереди, а пропустить задание, поступившее позже остальных.
Т.е. на 1 уровне происходит фильтрация заданий, но ресурсы к ним не определяются. В задании должна быть указана программа и данные. В результате, имеем очередь входных заданий.
2 уровень. Если в системе есть свободные ресурсы (см. определение выше), то всплывает планировщик второго уровня. Он выбирает из очереди задание, определяет, можно ли его решить с помощью ресурсов, что есть в системе. Если можно, то вызывает инициализатор/инициатор, который как только определит ресурсы и расшифрует задание, сформирует блок управления задачей/процессом (PCB).
Так исторически сложилось, что сначала был TCB, а потом появился программный режим работы, и появились процессы и PCB.

Таким образом, задание превращается в задачу/процесс. Возможна ситуация, когда процессов слишком много и они все в памяти не помещаются, тогда некоторые из них будут выгружены на диск. Второй уровень планирования определяет, какие процессы можно хранить в памяти, а какие — на диске. Этим занимается планировщик памяти.
Для оптимизации эффективности системы планировщик памяти должен ре шить, сколько и каких процессов может одновременно находиться в памяти. Кол-во процессов, одновременно находящихся в памяти, называется степенью многозадачности.
Планировщик памяти периодически просматривает процессы, находящиеся на диске, чтобы решить, какой из них переместить в память. Среди критериев, используемых планировщиком, есть следующие:

1. Сколько времени прошло с тех пор, как процесс был выгружен на диск или загружен с диска?
2. Сколько времени процесс уже использовал процессор?
3. Каков размер процесса (маленькие процессы не мешают)?
4. Какова важность процесса?

3 уровень. Как только процессор освобожден (либо естественно либо с вытеснением), срабатывает планировщик 3-го (верхнего уровня, он же планировщик процессора) и из готовых процессов/задач он должен выбрать процесс/задачу для выполнения и занять время выполнение в процессоре.
4 уровень. Собственно, его можно не выделять отдельно. На последнем этапе результаты выполнения могут быть сохранены или выведены на внешний носитель.

3. Страничная организация памяти. Виды. Особенности.

В основе **виртуальной памяти** лежит идея, что у каждой программы имеется свое собственное адресное пространство, которое разбивается на участки, называемые страницами. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы присутствие в памяти всех страниц не обязательно. Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету. Когда программа ссылается на часть своего адресного пространства, которое не находится в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду. Страничная организация подразумевает разбиение адресного пространства на страницы фиксированного размера и разбиения ОП на фреймы того же размера.
Виды страничной организации:
А)Чистая страничная – вся програма находится в памяти, грузится сразу с использованием разбиения на страницы (фреймы). Чем меньше размер страницы тем меньше фрагментация, но сразу растет таблица страниц. Различие между сегм. и странич. организацией не надо складывать адр. стран. и смещение, достаточно конкатенации. Таблица страниц = кол-ву страниц в програме. При формировании испол. адреса, если бит присутствия = 0, то проис. разрыв страницы, т.е. подгруж страница (по прерыванию).
Б)Страничная по запросу - чисто страничная организация памяти подразумевает, что все страницы программы находятся в ОП. При страничной организации по запросам в таблице страниц появляется бит присутствия и страницы грузятся в ОП по мере необходимости. При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная страница находится в оперативной памяти то производится обращение. Если же нужная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание.
В)Сегм.-стран организация. При сегментно-страничной организации программа разбита на блоки, которые вызываются по мере необходимости, а для каждого блока своя таблица страниц. Программист работает на уровне сегментов.

4. Особенности управления файлами в ОС UNIX. Отличия. Достоинства, недостатки.

Первый блок — это ***суперблок*** (superblock), в котором хранится информация о компоновке файловой системы, включая количество i-узлов, количество дисковых блоков, начало списка свободных дисковых блоков (это обычно несколько сотен элементов).
Суперблок файловой системы - содержит оперативную информацию о состоянии файловой системы, а также данные о параметрах настройки файловой системы. В частности суперблок имеет информацию о:

- количестве индексных дескрипторов (ИД) в файловой системе;
- размере файловой системы;
- свободных блоках файлов;
- свободных ИД;
- еще ряд данных, которые мы не будем перечислять в силу уникальности их назначения

Информация о суперблоке:

- суперблок всегда находится в ОП; (Недостаток)
- все операции по освобождению блоков, занятию блоков файлов, по занятию и освобождению ИД происходят в ОП (минимизация обменов с диском). Если же содержимое суперблока не записать на диск и выключить питание, то возникнут проблемы (несоответствие реального состояния файловой системы и содержимого суперблока). Но это уже

требование к надежности аппаратуры системы. Свойство файловой системы по оптимизации доступа, критерием которого является количество обменов, которые файловая система производит для своих нужд, не
связанных с чтением или записью информации файлов (Преимущество).

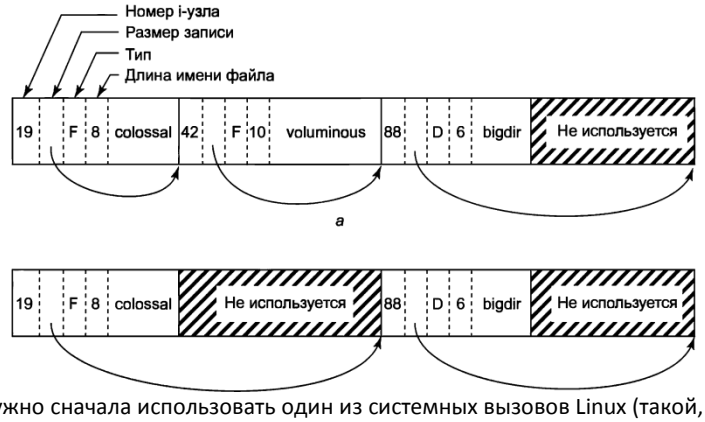
Индексные Дескрипторы — это объект Unix, который ставится во взаимнооднозначное соответствие с содержимым файла. То есть для каждого ИД существует только одно содержимое и наоборот, за исключением лишь той ситуации, когда файл ассоциирован с каким-либо внешним устройством. Напомним содержимое ИД:

- поле, определяющее тип файла (каталоги и все остальные файлы);
- код привилегии/защиты;
- количество ссылок к данному ИД из всевозможных каталогов файловой системы;
- (нулевое значение означает свободу ИД)
- длина файла в байтах;
- даты и времена (время последней записи, дата создания и т.д.);
- поле адресации блоков файла.

Как видно — в ИД нет имени файла. Давайте посмотрим, как организована адресация блоков, в которых размещается файл. В поле адресации находятся номера первых десяти блоков файла, то есть если файл небольшой, то вся информация о размещении данных файла находится непосредственно в ИД. Если файл превышает десять блоков, то начинает работать некая списочная структура, а именно, 11й элемент поля адресации содержит номер блока из пространства блоков файлов, в которых размещены 128 ссылок на блоки данного файла. В том случае, если файл еще больше — то используется 12й элемент поля адресации. Сутьего в следующем — он содержит номер блока, в котором содержится 128 записей о номерах блоках, где каждый блок содержит 128 номеров блоков файловой системы. А если файл еще больше, то используется 13 элемент — где глубина вложенности списка увеличена еще на единицу.

Таким образом мы можем получить файл размером (10+128+1282+1283)*512.

Затем следует дескриптор группы, содержащий информацию о расположении битовых массивов, количестве свободных блоков и i-узлов в группе, а также о количестве каталогов в группе. Эта информация важна, так как файловая система ext2 пытается распределить каталоги равномерно по всему диску. В двух битовых массивах ведется учет свободных блоков и свободных i-узлов (это тоже унаследовано из файловой системы MINIX 1 и отличается от большинства файловых систем UNIX, в которых для свободных используется список). Размер каждого битового массива равен одному блоку. При размере блока в 1 Кбайт такая схема ограничивает размер группы блоков 8192 блоками и 8192 i-узлами. Первое число является реальным ограничением, а второе — практически нет. Затем располагаются сами i-узлы. Они нумеруются от 1 до некоторого максимума. Размер каждого i-узла — 128 байт, и описывает он ровно один файл, i-узел содержит учетную информацию (в том числе всю возвращаемую вызовом stat> который просто берет ее из i-узла), а также достаточное количество информации для определения местоположения всех дисковых блоков, которые содержат данные файла. Следом за i-узлами идут блоки данных. Здесь хранятся все файлы и каталоги. Если файл или каталог состоит более чем из одного блока, то эти блоки не обязаны быть непрерывными на диске. В действительности блоки большого файла, скорее всего, будут разбросаны по всему диску. Соответствующие каталогам i-узлы разбросаны по всем группам дисковых блоков. Ext2 пытается расположить обычные файлы в той же самой группе блоков, что и родительский каталог, а файлы данных в том же блоке, что и i-узел исходного файла (при условии, что там имеется достаточно места). Битовые массивы используются для того, чтобы принимать быстрые решения относительно выделения места для новых данных файловой системы. Когда выделяются новые блоки файлов, то ext2 также делает упреждающее выделение (preallocates) нескольких (восьми) дополнительных блоков для этого же файла (чтобы минимизировать фрагментацию файла из-за будущих операций записи). Эта схема распределяет файловую систему по всему диску. Она также имеет хорошую производительность (благодаря ее тенденции к смежному расположению и пониженной фрагментации).



Для доступа к файлу нужно сначала использовать один из системных вызовов Linux (такой, как open), для которого нужно указать путь к файлу. Этот путь разбирается, и из него извлекаются составляющие его каталоги. Если указан относительный путь, то поиск начинается с текущего каталога процесса, в противном случае — с корневого каталога. В любом случае, i-узел для первого каталога найти легко: в дескрипторе процесса есть указатель на него либо (в случае корневого каталога) он обычно хранится в определенном блоке на диске. Каталог позволяет использовать имена файлов до 255 символов. Каждый каталог состоит из некоторого количества дисковых блоков (чтобы каталог можно было записать на диск атомарно). В каталоге элементы для файлов и каталогов находятся в несортированном порядке (каждый элемент непосредственно следует за предыдущим). Элементы не могут пересекать границы блоков, поэтому в конце каждого дискового блока обычно имеется некоторое количество неиспользуемых байтов.

Файловая система представляет собой дерево, начинающееся в корневом каталоге, с добавлением связей, формирующих направленный ациклический граф. Имена файлов могут содержать до 14 любых символов ASCII, кроме слэша (поскольку он служит разделителем компонентов пути) и символа NUL (поскольку он используется для дополнения имен короче 14 символов). Символ NUL имеет числовое значение 0.

Для учета дисковых блоков файла используется общий принцип позволяющий работать с очень большими файлами. Первые 10 дисковых адресов хранятся в самом i-узле, поэтому для небольших файлов вся необходимая информация содержится непосредственно в i-узле, считываемом с диска в оперативную память при открытии файла. Для файлов большего размера один из адресов в i-узле представляет собой адрес блока диска, называемого однократным косвенным блоком. Этот блок содержит дополнительные дисковые адреса. Если и этого недостаточно, используется другой адрес в i-узле, называемый двукратным косвенным блоком и содержащий адрес блока, в котором хранятся адреса однократных косвенных блоков. Если и этого мало, используется трехкратный косвенный блок.

5. Прерывание — основа управления процессами в ОС.

Прерывание — один из основных механизмов организации многопрограммных режимов работы и обмена информацией с внешними устройствами.

Практически любая вычислительная система, кроме одного или нескольких процессоров имеет, в своем составе множество устройств, обрабатывающих данные параллельно с процессором, что повышает общую производительность системы. Это внешние накопители, устройства коммутации, дополнительные специализированные процессоры, таймеры и прочие устройства. При такой организации системы возникает проблема реализации взаимодействия между процессором и периферийными устройствами.

Взаимодействие с внешними устройствами

Рассмотрим такое взаимодействие подробнее. Устройство может независимо от процессора выполнять операции ввода/вывода, а иногда и помещать результат в оперативную память (используя прямой доступ к памяти). Если процессору необходимо принять или передать данные при помощи внешнего устройства, он посылает устройству запрос на выполнение определенных действий. Если устройство готово его обработать, оно выполняет программу ввода/вывода и удовлетворяет запрос, а процессор получает результат его работы. Таким образом, процессору необходимо активизировать устройство, назначить ему работу и получить результат (либо информацию о завершении обработки). Рассмотрим возможные варианты организации такого обмена:

- Процессор передает устройству необходимую для работы информацию, а затем постоянно проверяет состояние устройства до появления в нем признака окончания обработки, после чего (если необходимо) принимает результат. Такой режим называют режимом непосредственного (программного) опроса (Polling mode).
- Процессор помещает необходимые устройству данные в определенное место, откуда устройство их считывает, затем выполняет необходимую работу и располагает результат также в определенном месте, из которого процессор сможет их получить. Это развитие режима непосредственного опроса, исключающее прямое взаимодействие процессора с устройством, получило название системы с почтовым ящиком (Mailbox system).
- Процессор посылает устройству запрос и выполняет процесс, не связанный с окончанием работы внешнего устройства. Устройство выполняет свои функции, а после формирования результата посылает процессору специальный сигнал, сообщающий о том, что обработка закончена. Процессор обращается к устройству и получает необходимую информацию. Именно такой метод реализуется при помощи системы прерываний (Interrupt system).

Эти методы различны по объему требуемых для их реализации ресурсов, по времени получения результата взаимодействия процессора и периферийного устройства и по временным затратам на ожидание взаимодействия. Так, режим непосредственного опроса требует минимального количества ресурсов и позволяет получить результат наиболее быстрым образом. Однако, на ожидание готовности устройства процессор тратит довольно много времени, особенно, если устройство имеет невысокую скорость обработки данных (для некоторых устройств ввода/вывода, например, для печатающих устройств, скорость обработки информации в тысячи и миллионы раз ниже, чем у процессора), а это значит, что во время взаимодействия процессор большую часть времени тратит зря.

Системы с "почтовыми ящиками" позволяют организовать более независимую работу процессора и внешнего устройства, но это увеличивает время получения результата и затраты на дополнительное оборудование.

Прерывания сводят время бесполезного ожидания к нулю, но реализация системы прерываний требует наибольших затрат оборудования, к тому же необходимость переключения выполняемой в процессоре задачи на программу взаимодействия с устройством требует дополнительного времени.

Именно поэтому, в вычислительных системах одновременно могут использоваться различные методы организации взаимодействия процессоров и вспомогательных устройств в зависимости от требований, предъявляемых к такому взаимодействию. Одни устройства могут использовать прерывания, другие — непосредственную связь с процессором или "почтовые ящики". Существуют также комбинации этих методов. Например, разновидностью метода непосредственного опроса является режим, в котором процессор, выполняя основную работу, прерывается сигналом таймера через определенные промежутки времени. При этом он проверяет состояние всех подключенных к нему устройств, отмечая произошедшие со времени последнего опроса изменения, и, если таковые присутствуют, проверяет, имеются ли в устройстве необходимые ему данные и принимает имеющуюся для него информацию. Завершив опрос, процессор продолжает выполнение прерванного процесса. Аналогичный вариант возможен и при просмотре "почтовых ящиков". Однако, наиболее часто используется механизм прерываний, особенно в многозадачных системах, так как с точки зрения затраченного времени этот режим наиболее экономный.

Механизм прерываний используется не только при взаимодействии процессора с внешними устройствами. Ведь в качестве ресурса могут выступать не только внешние устройства, но и процессор по отношению к процессам, а также операционная система по отношению к программе пользователя. Поэтому прерывания являются основой функционирования ОС (особенно многозадачной).

Обработка прерываний

В общем случае под прерыванием понимают событие, требующее от системы прекращения работы активного процесса и перехода к обработке другого задания. Вычислительная система, обладающая возможностью использования прерываний, должна содержать целый комплекс программно-аппаратных средств, обеспечивающих регистрацию сигнала прерывания, приостановку активного процесса, переход к обработке прерывания и восстановление прерванного процесса. Совокупность действий, выполняемых после регистрации сигнала прерывания, называется обработкой прерывания. Система обработки прерываний — комплекс программно-аппаратных средств, обеспечивающих приостановку активного процесса и переход к обработке прерывания.

Рассмотрим работу системы прерываний подробней. В реальной жизни нам очень часто приходится сталкиваться с аналогами прерываний, реализованных в вычислительных системах. К примеру, предположим, что вы читаете книгу, и в это время раздается телефонный звонок. Тогда вы откладываете книгу, подходите к телефону, отвечаете на звонок, а затем возвращаетесь к прерванному на самом интересном месте занятию. В этот момент вам необходимо определить, с какого места продолжить чтение. Вернемся к тому моменту, когда вы отложили книгу. Если вы ожидали важного звонка, то вы могли закрыть книгу, даже не запомнив места, на котором остановились. В другом случае вы могли бы оставить пометку карандашом в том месте, где вас прервал телефонный звонок, а на нужной странице положить закладку. Но если книга вас весьма заинтересовала, то вы могли бы не сразу реагировать на звонок, а дочитать до конца предложения, абзаца, а то и до конца "самого интересного места".

Аналогичным образом работают прерывания в вычислительных системах. Когда процессор получает сигнал прерывания, он должен прервать выполняемый в нем процесс. При этом возможны несколько вариантов реакции процессора на сигнал прерывания:

1. Прерывание возможно только после определенных команд, при этом необходимо сохранение минимального количества информации о состоянии системы.
2. Прерывание возможно после завершения любой команды процессора.
3. Прерывание возможно после завершения очередного такта процессора. При этом требуется сохранение большого объема информации.

Большинство процессоров построены таким образом, что сигнал прерывания проверяется ими только после завершения выполнения текущей команды выполняющегося процесса. Однако, в системе могут существовать прерывания, для которых невозможно ожидание завершения очередной команды, например, если возникает ошибка в самой команде. Поэтому в системе могут использоваться одновременно несколько из перечисленных способов для обработки различных прерываний.

Обнаружив сигнал прерывания, процессор должен запомнить состояние прерванного процесса для того, чтобы продолжить его выполнение после обработки прерывания. После этого в процессор загружается новый процесс, выполняющий обработку прерывания. Эта процедура получила название **переключения контекста** или **переключения состояния**. Она является одной из важнейших в организации работы операционной системы, причем ее реализация требует поддержки и выполнения некоторых функций на аппаратном уровне.

Каждый процессор оснащен аппаратурой для управления последовательностью выполнения команд. Это специальный регистр, постоянно хранящий адрес следующей команды. Он называется **словом состояния программы** *(Program Status Word, PSW)*, иначе называемый также **словом состояния процесса** *(Process Status Word, PSW)*, **счетчиком программы** *(Program Counter, PC),* **счетчиком адреса программы** *(Program Address Counter, PAC)*, **указателем команды** *(Instruction Pointer, IP)*. Как минимум, такой регистр должен содержать адрес следующей команды (или последовательности команд), а в дополнение к этому — различную информацию, которую система связывает с процессом.

При переключении контекста процессор сохраняет содержимое PSW, затем помещает в него новое значение, соответствующее процессу, называемому **обработчиком прерывания** *(Interrupt Handler, IH)*, который и выполняет обработку самого прерывания. После завершения обработки прерывания в PSW восстанавливается содержимое сохраненного ранее PSW.

В процессе обработки прерывания можно выделить следующие **фазы прерывания**:

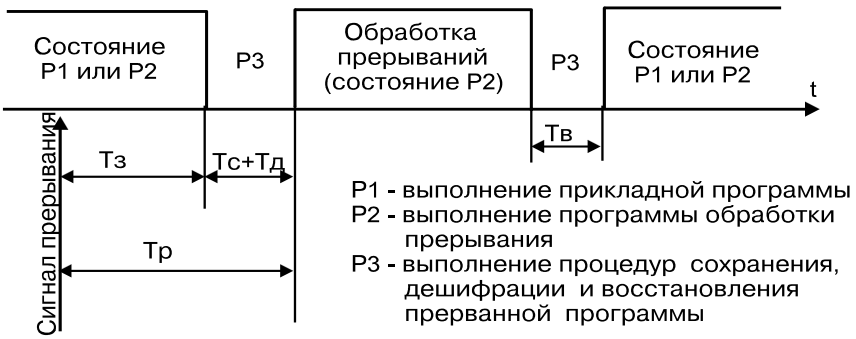


Рис. 2.20. Фазы прерывания

1. $T_з$ — **время задержки** между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания (смотри выше).
2. $T_с$ — **время сохранения** необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.
3. $T_д$ — **время дешифрации** сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.
4. $T_в$ — **время восстановления** прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется **временем реакции системы на сигнал прерывания** ($T_р$).

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях **запросов на прерывания.** Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание.

Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

1. Полноупорядоченная схема**.** Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить *все* регистры запросов на прерывания с более высоким приоритетом.
2. Частично упорядоченная схема**.** Все прерывания делятся на **классы** и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.

Классы прерываний

По своему назначению, причине возникновения прерывания делятся на различные **классы**. Традиционно выделяют следующие классы:

1. *Прерывания от схем контроля машины*. Возникают при обнаружении сбоев в работе аппаратуры, например, при несовпадении четности в микросхемах памяти.
2. *Внешние прерывания*. Возбуждаются сигналами запросов на прерывание от различных внешних устройств: таймера, клавиатуры, другого процессора и пр.
3. *Прерывания по вводу/выводу*. Инициируются аппаратурой ввода/вывода при изменении состояния каналов ввода/вывода, а также при завершении операций ввода/вывода.
4. *Прерывания по обращению к супервизору*. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.
5. *Программные прерывания*. Возникают при выполнении **команды вызова прерывания** либо при обнаружении ошибки в выполняемой команде, например, при арифметическом переполнении.

В последнее время принято прерывания 4 и 5 классов объединять в один класс программных прерываний, причем, в зависимости от источника, вызвавшего прерывание, среди них выделяют такие подтипы:

- прерывание вызванное исполнением процессором *команды перехода к подпрограмме обработки прерывания*
- прерывания, возникающие в результате *исключительной* (аварийной) *ситуации* в процессоре (деление на "0", переполнение и т.д.).

В связи с многообразием различных ВС и их постоянным развитием меняется и организация системы прерываний. Так, с появлением виртуальной памяти, появился класс страничных прерываний, который можно отнести и к классу исключительных ситуаций в процессоре; в системах с кэш-памятью существуют прерывания подкачки страниц в кэш-память и т.д.

Во время выполнения обработчика одного из прерываний возможно поступление другого сигнала прерывания, поэтому система должна использовать определенную *дисциплину обслуживания заявок* на прерывания. Обычно различным классам либо отдельным прерываниям присваиваются различные *абсолютные приоритеты*, т.о. при поступлении запроса с высшим приоритетом текущий обработчик снимается, а его место занимает обработчик вновь поступившего прерывания. Количество уровней вложенности прерываний называют **глубиной системы прерываний**. Обработчики прерываний либо дисциплина обслуживания заявок на прерывание должны иметь также специальные средства для случая, когда при обработке прерывания возникает запрос на обработку прерывания от того же источника.