1. **Место операционной системы.**

**Операционная система** — это комплекс программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Большинство компьютеров имеют два режима работы:

* Режим ядра – Операционная система
* Режим пользователя – Программы пользовательского интерфейса и Прикладные программы

Важное отличие ОС от обычного ПО состоит в следующем:

* Заменимость
* Местоположение
* Довольно большой объем
* Сложная структура
* Длительные сроки использования

1. **Абстракция аппаратного обеспечения.**

**Абстракция аппаратного обеспечения** - наборы подпрограмм в ПО, которые предоставляют программам доступ к аппаратным ресурсам через программные интерфейсы.

**Драйвер** — это ПО, с помощью которого другое ПО (операционная система) получает доступ к аппаратному обеспечению некоторого устройства.

ОС содержат множество драйверов для управления устройствами. Поэтому все ОС предоставляют еще один уровень абстракции, к примеру, для использования дисков – файлы. Используя эту абстракцию, программы могут работать с файлами, не вникая в подробности реальной работы оборудования.

Хорошая абстракция разделяет сложную задачу на две: первая из этих задач состоит в определении и реализации абстракций, а вторая – в использовании этих абстракций для решения текущей проблемы.



1. **Управление ресурсами.**

Управление ресурсами включает в себя мультиплексирование (распределение) ресурсов двумя способами: во времени и в пространстве.

* Когда ресурс разделяется во времени, программы и пользователи используют его по очереди.
* Каждый клиент получает какую-то часть разделяемого ресурса.

1. **История операционных систем.**

История операционных систем включает следующие этапы:

1. Нулевое поколение (1940-е годы). Операционных систем не существует, все программы пишутся в машинных командах.
2. Первое поколение (1950-е годы). Основная задача ОС 50-х годов — упрощение перехода с задачи на задачу.
3. Второе поколение (начало 1960-х годов). Стали разрабатываться ОС с более широкими возможностями.
4. Третье поколение (середина 1960-х — середина 1970-х). Операционные системы третьего поколения стали программной прослойкой между пользователями и аппаратурой ЭВМ.
5. Четвертое поколение (середина 1970-х годов — 90-е). Появление четвертого поколения ОС связано с распространением вычислительных сетей и появлением микропроцессора и персонального компьютера.

К началу 90-х годов практически все ОС стали сетевыми. Появились специализированные ОС, которые предназначены исключительно для выполнения коммуникационных задач.

1. **Процессоры.**

**Процессор** — это основной вычислительный элемент и «мозг» компьютера, от которого во многом зависит скорость его работы и производительность.

Для каждого типа процессора существует определенный набор команд, которые он может выполнить.

**Вектор прерывания** – сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего внимания.

1. **Основные регистры процессора.**

**Регистр** **процессора** - **это** быстро доступная ячейка, доступная процессору компьютера.

Поскольку доступ к памяти для получения команды или данных занимает намного больше времени, чем выполнение команды, у всех ЦП есть несколько собственных регистров для хранения основных переменных и промежуточных результатов.

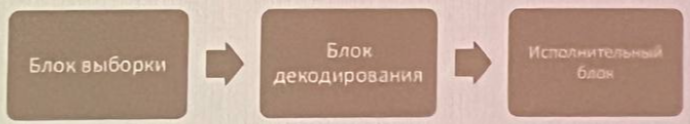
В дополнение к регистрам общего назначения, у многих процессоров есть ряд специальных регистров, доступных программисту.

* Счетчик команд
* Указатель стека
* Слово состояния программы – PSW (Program Status Word)

1. **Организация работы процессоров.**

Есть два вида:

**Конвейер** – выполнение более одной команды одновременно.



**Суперскалярный** процессор имеет несколько исполнительных блоков. Одновременно выбираются две и более команды, которые декодируются и помещаются в буфер хранения, в котором ожидают возможности своего выполнения.



1. **Многопоточные и многоядерные микропроцессоры.**

**Многопоточные микропроцессоры** – процессоры, которые обладают технологией многопоточности.

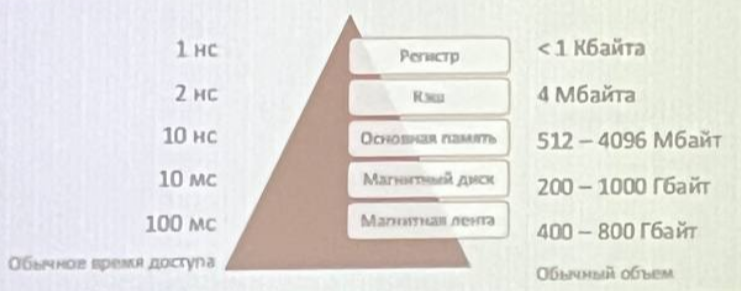
**Многопоточность** – дублирование функциональных блоков и части управляющей логики.

**Многоядерные процессоры** – процессоры, имеющие на одном кристалле четыре, восемь и более полноценных процессоров, или ядер.

1. **Память.**

**Память** - часть вычислительной машины, физическое устройство или среда для хранения данных, используемая в вычислениях в течение определённого времени.

К памяти относятся: Регистр, Кэш, Основная память, Магнитный диск и Магнитная лента.



Верхние уровни обладают более высоким быстродействием, меньшим объемом и более высокой удельной стоимостью хранения одного бита информации, чем ниже уровни.

1. **Назначение кэш-памяти.**

**Кэш**-**память** устройства — это особая форма памяти, которая используется для временного хранения данных, чтобы ускорить их загрузку. **Кэш**-**память** размещается между более медленными хранилищами данных, такими как оперативная память и жесткий диск, и более скоростными компонентами системы, обеспечивая быстрый доступ к наиболее часто используемым данным.

Оперативная память разделяется на **кэш-строки** обычно по 64 байта. Наиболее интенсивно используемые кэш-строки оперативной памяти сохраняются в высокоскоростной кэш-памяти, находящейся внутри процессора, или очень близко к нему.

1. **Кэш L1 и L2.**

Современные процессоры имеют сразу два уровня кэш памяти.

* Первый уровень, или **кэш L1**, всегда является частью самого процессора и обычно подает декодированные команды в процессорный механизм исполнения команд. Обычно каждый из кэшей L1 имеет объем 16 Кбайт.
* Второй уровень, или **кэш L2** содержит несколько мегабайт недавно использованных слоев памяти.

1. **Типы операционных систем: Операционные системы мейнфреймов, Серверные операционные системы, Многопроцессорные операционные системы и Операционные системы персональных компьютеров.**

* **ОС мейнфреймов** ориентированы на одновременную обработку множества заданий, большинство из которых требует колоссальныхобъемов ввода-вывода данных.Обычно они предлагают три вида обслуживания: пакетную обработку, обработку транзакций и работу в режиме разделения времени.

Примеры: OS/390, Linux.

* **Серверные ОС** одновременно обслуживают по сети множество пользователей, предоставляя им общий доступ к аппаратным и программным ресурсам. Серверы могут предоставлять услуги печати, хранения файлов или веб-служб.

Примеры: Solaris, FreeBSD, Linux.

* **Многопроцессорные ОС** - это системы, состоящие из двух и более процессоров общего назначения, которые осуществляют параллельное выполнение команд.

Примеры: Linux, Solaris, Windows NT.

* **ОС персональных компьютеров** рассчитаны на персональные компьютеры и ноутбуки.

Примеры: Windows, MacOS, Linux.

1. **Типы операционных систем: Мобильные операционные системы, Встроенные операционные системы, Операционные системы сенсорных узлов, Операционные системы реального времени и Операционные системы смарт-карт.**

* **Мобильные ОС** предназначены для смартфонов, планшетов и других мобильных устройств.

Примеры: Android, IOS.

* **Встроенные ОС** работают на компьютерах, которые управляют различными устройствами.

Примеры: Embedded Linux, QNX, VxWorks.

* **ОС сенсорных узлов** – сети, составленные из миниатюрных сенсорных узлов, связанных друг с другом и с базовой станцией по беспроводным каналам. Узлы такой сети представляют собой миниатюрные компьютеры, питающиеся от батареи и имеющие встроенную радиосистему.

Примеры: TinyOS.

* **ОС реального времени** - это ОС для вычислительных приложений реального времени, которая обрабатывает данные и события, имеющие критически определенные временные ограничения. Бывают системы жесткого (не допустима задержка) и мягкого (допустима задержка) реального времени.

Примеры: eCos.

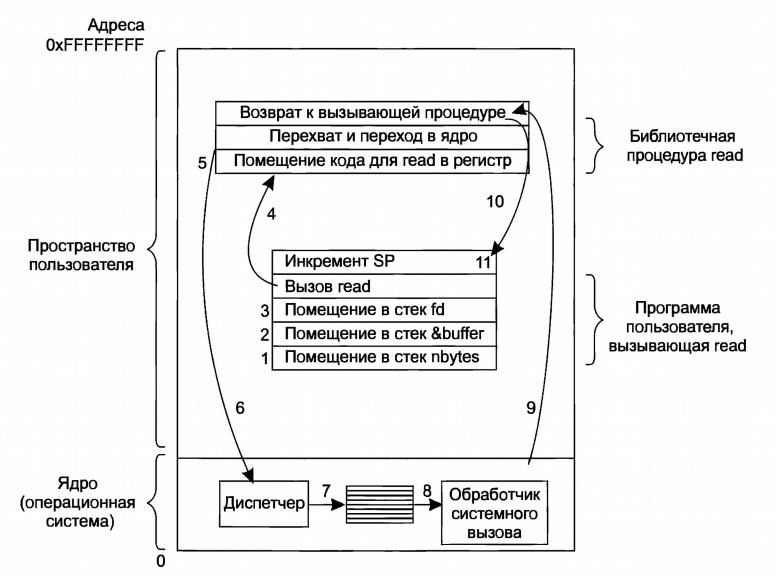
* **ОС смарт-карт** самые маленькие ОС. На них накладываются очень жесткие ограничения по требуемой вычислительной мощности процессора и объему памяти.

1. **Системные вызовы.**

**Системный вызов** – обращение прикладной программы к ядру ОС для выполнения какой-либо операции. Существуют системные вызовы для:

* Управления процессами (fork(), exec(), wait(), kill());
* Управления файлами (open(), close(), read(), write());
* Управления каталогами (mkdir(), link(), unlink(), mount(), unmount());
* Другие (chdir(), chmod()).

1. **Пример выполнения системного вызова read.**



11 этапов:

* Шаг 1-3, вызывающая программа помещает параметры в стек.
* Шаг 4, вызов библиотечной процедуры.
* Шаг 5, помещение номера системного вызова в регистр.
* Шаг 6, выполнение с фиксированного адреса.
* Шаг 7, проверка номера системного вызова и передача его обработчику.
* Шаг 8, обработка системного вызова.
* Шаг 9, возврат управления библиотечной процедуре.
* Шаг 10, процедура возвращает управление пользовательской программе.
* Шаг 11, очистка стека.

1. **Windows Win32 API.**

**Win32 API** – набор процедур, определенный корпорацией Microsoft. Программисты должны использовать его для доступа к службам ОС.

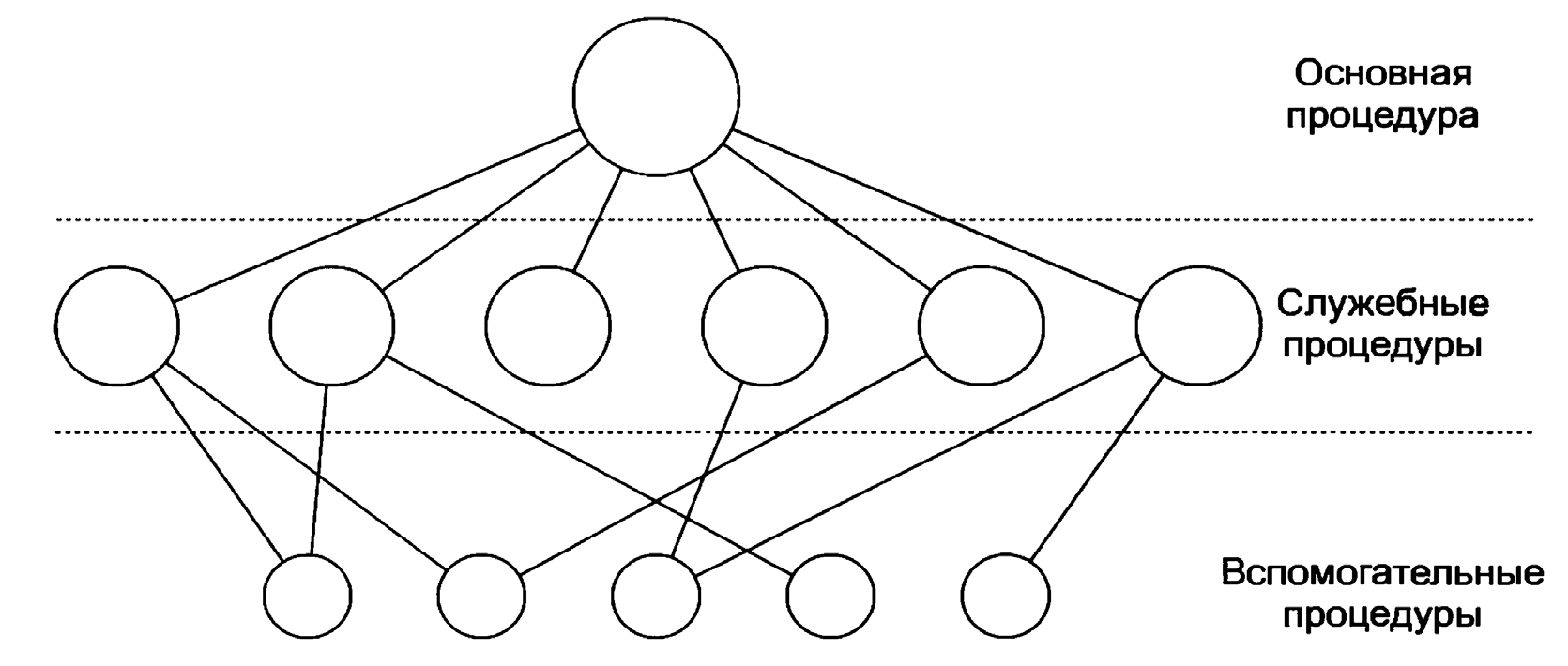


1. **Структура операционной системы - Монолитные системы.**

Вся ОС работает как единая программа в режиме ядра. ОС написана в виде процедур, связанных 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.

Базовая структура монолитной ОС:

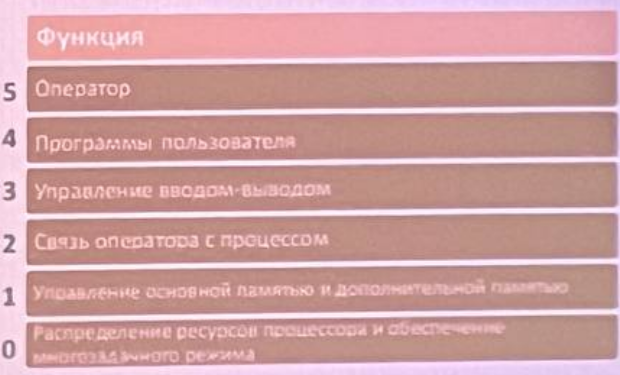
1. Основная программа, которая вызывает требуемую служебную процедуру.
2. Набор служебных процедур, выполняющих системные вызовы.
3. Набор вспомогательных процедур, содействующих работе служебных процедур.



1. **Структура операционной системы - Многоуровневые системы.**

Организация ОС в виде иерархии уровней, каждый из которых является надстройкой над нижележащими уровнями.

Первой такой системой была система THE.



1. **Структура операционной системы - Микроядра.**

Замысел, направлен на достижение высокой надежности за счет разбиения ОС 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.

Идея заключается в том, чтобы помещать в ядра исполнительный механизм, а не политику.

1. **Структура операционной системы - Клиент-серверная модель.**

Небольшая вариация идеи микроядер выражается в обособлении двух классов процессов: серверов, представляющих какую-нибудь службу, и клиентов, которые пользуются этими службами.

1. **Процессы.**

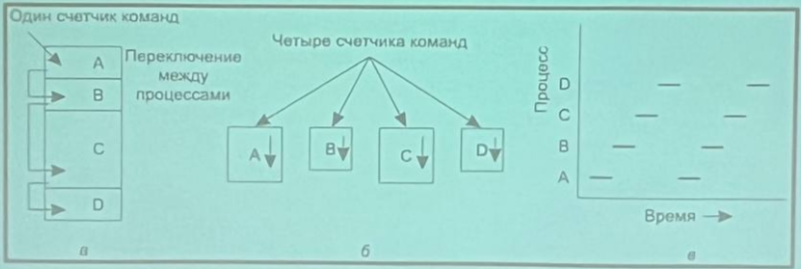
**Процесс** – экземпляр программы, выполняющейся в ОС.

В любой многозадачной системе ЦП быстро переключается между процессами, предоставляя каждому из них десятки или сотни миллисекунд. В каждый конкретный момент времени ЦП работает только с одним процессом, в течение 1 секунды он может успеть поработать с несколькими из них, создавая иллюзию параллельной работы. В этом случае говорят о псевдопараллелизме в отличие от настоящего аппаратного параллелизма в многопроцессорных системах.

1. **Модель процесса.**

**Процесс** – экземпляр программы, включая текущие значения счетчика команд, регистров и переменных.

ЦП постоянно переключается между процессами, такое переключение между процессами, называется мультипрограммированием, или многозадачным режимом работы.



1. **Создание процесса.**

Существует четыре основных события, приводящих к созданию процессов.

1. Инициализация системы.
2. Выполнение работающим процессом системного вызова, предназначенного для создания процесса.
3. Запрос пользователя на создание нового процесса.
4. Инициация пакетного задания.
5. **Завершение процесса.**

Процессы завершаются в силу следующих обстоятельств:

* Обычный выход (добровольно);
* Выход при возникновении ошибки (добровольно);
* Возникновение фатальной ошибки (принудительно);
* Уничтожение другим процессом (принудительно);

1. **Иерархии процессов.**

Когда процесс порождает другой процесс, родительский и дочерний процесс продолжают оставаться определенным образом связанными друг с другом. Дочерний процесс может и сам создать какие-нибудь процессы, формируя иерархию процессов.

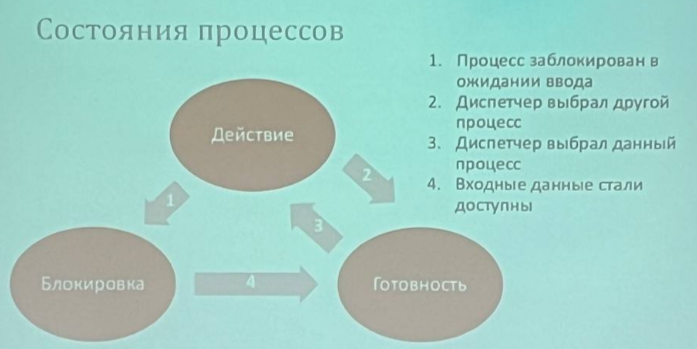
В UNIX процесс, все его дочерние процессы и более отдаленные потомки образуют группу процессов.

В Windows все процессы являются равнозначными. Происходит присвоение родительскому процессу **дескриптора**, который может им использоваться для управления дочерним процессом.

1. **Состояния процессов.**

Состояния, в которых может находиться процесс:

* Выполняемый (в данный момент использующий ЦП);
* Готовый (работоспособный, но временно приостановленный, чтобы дать возможность выполнения другому процессу);
* Заблокированный (неспособный выполняться, пока не возникнет какое-нибудь внешние событие).



1. **Реализация процессов.**

Для реализации модели процессов ОС ведет таблицу, называемую **таблицей процессов** (блоками управления процессами), в которой каждая запись соответствует какому-нибудь процессу.

Эти записи содержат важную информацию о состоянии процесса, что должно быть сохранено, когда процесс переключается из состояния *выполнения* в состояние *готовности* и *блокировки*, что бы позже он мог возобновить выполнение, как будто никогда не останавливался.

1. **Применение потоков.**

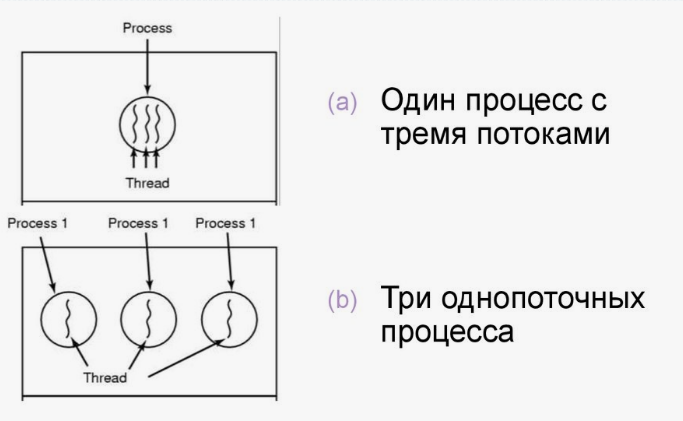
**Поток** – легковесный процесс, который выполняется в рамках другого процесса. Потоки могут использовать адресное пространство процесса, в котором они выполняются.

* Основная причина использования потоков заключается в том, что во многих приложениях одновременно происходит несколько действий, часть из которых может быть заблокированной.
* Вторым аргументом в пользу потоков является легкость их создания и ликвидации по сравнению с процессами.
* Третий аргумент, когда выполнятся значительные вычисления, а также значительная часть времени тратится на ожидание ввода-вывода, наличие потоков позволяет этим действиям перекрываться по времени, ускоряя работу приложения.

1. **Классическая модель потоков.**

У потока есть:

* Счетчик команд, отслеживающий, какую очередную инструкцию нужно выполнять.
* Регистры, в которых содержатся текущие рабочие переменные.
* Стек с протоколом выполнения, содержащим по одному фрейму для каждой вызванной, но еще не возвратившей управление процедуры.
* Состояние потока (выполняемый, заблокированный, готовый и завершенный).



1. **Потоки в POSIX.**

Чтобы определить возможность создания переносимых многопоточных программ, в отношении потоков был определен стандарт IEEE standard 1003.1c. Определенный в нем пакет, касающийся потоков, называется Pthreads. Он поддерживается большинством UNIX-систем.

Примеры вызовов:

pthread\_create, pthread\_exit, pthread\_join, pthread\_yield.

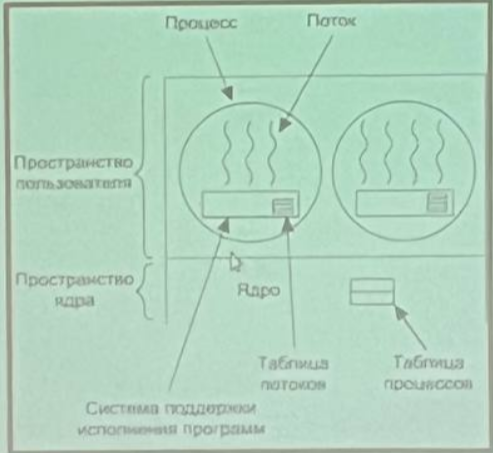
1. **Реализация потоков в пользовательском пространстве.**

Весь набор потоков помещается в пользовательское пространство. И об этом наборе ядру ничего не известно.

* Первое и самое очевидное преимущество состоит в том, что набор потоков на пользовательском уровне может быть реализован в ОС, которая не поддерживает потоки. При этом подходе потоки реализованы с помощью библиотеки.
* Переключение потоков происходит быстрее, чем перехват управления ядром, что является веским аргументом в пользу набора потоков, реализуемого на пользовательском уровне.
* Процедура, которая сохраняет состояние потока, и планировщик – это всего лишь локальные процедуры, поэтому их вызов намного более эффективен, чем вызов ядра.
* Такая реализация позволяет каждому процессу иметь собственные настройки алгоритма планирования.
* Эти потоки также лучше масштабируются, поскольку потоки в памяти ядра, безусловно, требуют в ядре пространства для таблицы и стека, что при очень большом количестве потоков может вызвать затруднения.

У потоков, реализованных на пользовательском уровне, есть существенные проблемы.

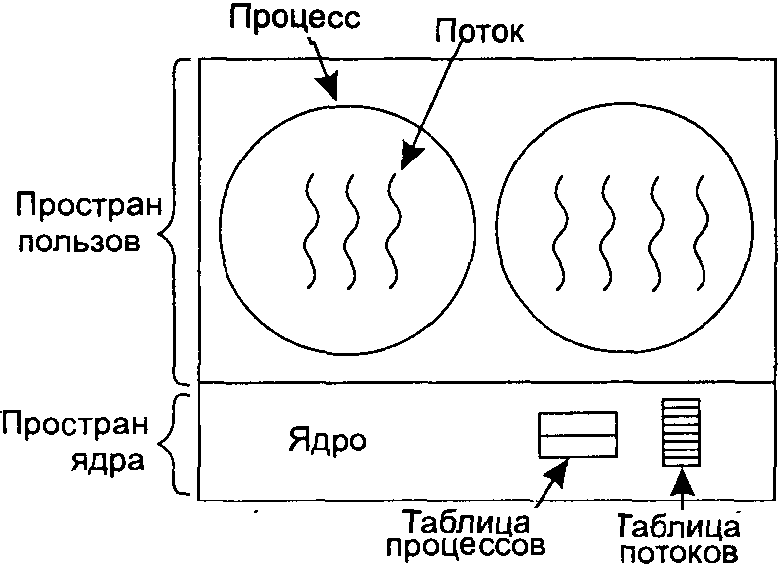
* Как реализовать блокирующие системные вызовы.
* Если начинается выполнение одного из потоков, то никакой другой поток, принадлежащий этому процессу, не сможет выполняться до тех пор, пока первый поток добровольно не уступит ЦП.



1. **Реализация потоков в ядре.**

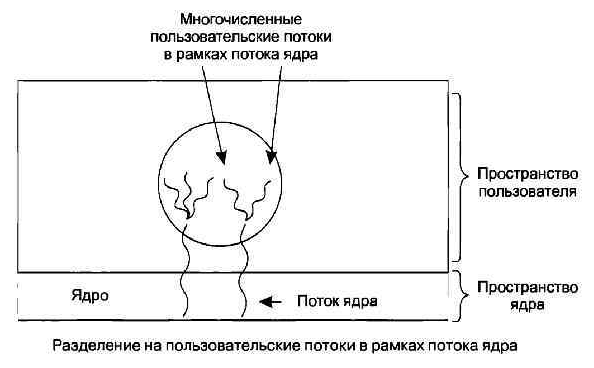
**Реализация потоков в ядре** предполагает, что ядро операционной системы владеет всей информацией о потоках и может управлять ими.

Все вызовы, способные заблокировать поток, реализованы как системные, с более существенными затратами, чем вызов процедуры в системе поддержки исполнения программ.



1. **Гибридная реализация.**

**Гибридная реализация** — это модель, в которой используется планирование потоков, как на уровне ядра, так и на уровне пользователя.



1. **Активация планировщика.**

Цель работы по **активации планировщика** заключается в имитации функциональных возможностей потоков на уровне ядра, но при лучшей производительности и более высокой гибкости, свойственной потокам, реализуемых в пользовательском пространстве.

Эффективность достигается путем уклонения от ненужных переходов между пространствами пользователя и ядра. Имеющаяся в пространстве пользователя система поддержки исполнения программ может заблокировать синхронизирующийся поток и самостоятельно спланировать работу другого потока.

1. **Всплывающие потоки.**

**Всплывающий поток** – поток, который создается системой по прибытии сообщения для его обработки. Основное преимущество заключается в том, что они создаются заново, нет никаких регистров, стека и всего остального, что должно быть восстановлено.

1. **Состязательная ситуация.**

**Состязательной** **ситуацией** называется ситуация, когда два или более процесса считывают или записывают какие-нибудь общие данные, а окончательный результат зависит от того, какой процесс и когда именно выполняется.

1. **Критические области.**

**Критическая область** – та часть программы, где используется доступ к общей памяти.

Чтобы параллельные процессы правильно выстраивали совместную работу и эффективно использовали общие данные. Для приемлемого решения необходимо соблюдение четырех условий:

1. Два процесса не могут одновременно находиться в критических областях.
2. Не должны выстраиваться никакие предположения по поводу скорости или количества ЦП.
3. Никакие процессы, выполняемые за пределами своих критических областей, не могут блокироваться любым другим процессом.
4. Процессы не должны находиться в вечном ожидании входа в свои критические области.
5. **Взаимное исключение с активным ожиданием.**

Постоянная проверка значения переменной, пока она не приобретет какое-нибудь значение, называется **активным ожиданием**. Как правило, этого ожидания следует избегать, поскольку оно тратит впустую время ЦП. Активное ожидание используется только в том случае, если есть основание полагать, что оно будет недолгим. Блокировка, использующая активное ожидание, называется **спин-блокировкой**.

* **Запрещение прерываний.** Запрет всех прерываний каждым процессом сразу после входа в критическую область и их разрешение сразу же после выхода из критической области.
* **Блокирующие переменные.** Использование одной общей переменной, исходное значение которой равно нулю. Когда процессу требуется войти в свою критическую область, сначала он проверяет значение блокирующей переменной.
* **Строгое чередование.**
* **Команда TSL.** TSL RX, LOCK

1. **Приостановка и активизация.**

Существуют примитивы взаимодействия процессов, которые блокируют работу, пока им не разрешается входить в критическую область, вместо напрасной траты времени ЦП.

Добавление **бита ожидания активизации**. Этот бит устанавливается, когда в отношении процесса, который не находится в состоянии бездействия, вызывается процедура wakeup. Затем когда процесс пытается заблокироваться при установленном бите ожидания активизации, этот бит снимается, но процесс не блокируется.

1. **Семафоры.**

**Семафор** – целочисленная переменная для подсчета количества активизаций, отложенных на будущее.

Проверка значения, его изменение, и, возможно, приостановка процесса осуществляется как единое и неделимое **атомарное действие**. Тем самым гарантируется, что с началом семафорной операции никакой другой процесс не может получить доступ к семафору до тех пор, пока не будет завершена или заблокирована.

Атомарность является абсолютно необходимым условием для решения проблем синхронизации и исключения состязательных ситуаций.

Используются две операции с семафорами:

* down – выясняет, отличается ли значение семафора от 0. Если отличается, она уменьшает это значение на 1 и продолжает свою работу. Если значение равно 0, процесс приостанавливается, не завершая в этот раз операцию down.
* up – увеличивает значение, адресуемое семафором, на 1. Если с этим семафором связаны один или более приостановленных процессов, способных завершить ранее начатые операции down, система выбирает один из них и позволяет ему завершить его операцию down. Таким образом, после применения операции up в отношении семафора, с которым были связаны приостановленные процессы, значение семафора так и останется нулевым, но количество приостановленных процессов уменьшится на 1.

1. **Мьютексы.**

**Мьютекс** – совместно используемая переменная, которая может находится в одном из двух состояний: заблокированном или незаблокированном.

Для работы с мьютиксами используются две процедуры.

Когда потоку (или процессу) необходим доступ к критической области, он вызывает процедуру mutex\_lock.

Если мьютекс уже заблокирован, вызывающий поток блокируется до тех пор, пока поток, находящийся в критической области, не завершит свою работу и не вызовет процедуру mutex\_unlock.

1. **Мониторы.**

**Монитор** – коллекция переменных и структур данных, сгруппированных вместе в специальную разновидность модуля или пакета процедур.

Процессы могут вызвать любые необходимые им процедуры, имеющиеся в мониторе, но не могут получить непосредственный доступ к внутренним структурам данных монитора из процедур, объявленных за пределами монитора.

В любой момент времени в мониторе может быть активен только один процесс.

Когда процедура монитора обнаруживает невозможность продолжения своей работы, она осуществляет операцию wait. Для активизации работы операцию signal.

1. **Передача сообщений.**

**Передача сообщений** – метод взаимодействия процессов, использует два примитива, send и receive, которые, подобно семафорам и в отличие от мониторов, является системными вызовами, а не конструкциями языка.

send(destination, &message);

receive(source, &message);

Проблемы разработки систем передачи сообщений.

* Утрата сообщений – **подтверждение**;
* Именование процессов – **аутентификации**;
* Производительность.

**Почтовый ящик** представляет собой место для буферизации конкретного количества сообщений, которое обычно указывается при его создании. При использовании почтовых ящиков в качестве параметров адреса в вызовах send и receive указываются почтовые ящики, а не процессы.

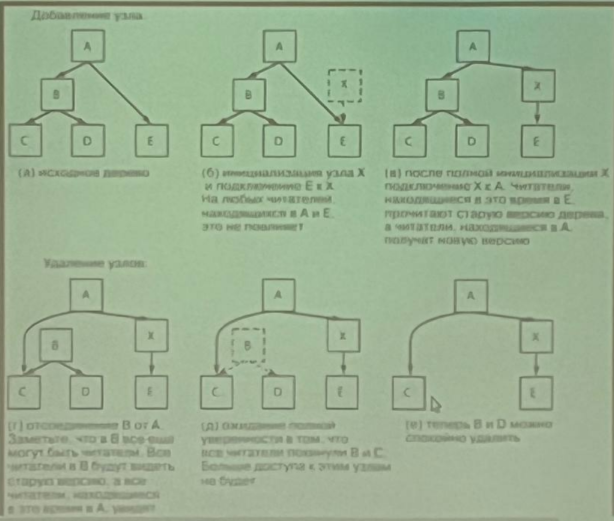
1. **Барьеры.**

Когда процесс достигает **барьера**, он блокируется до тех пор, пока этого барьера не достигнут все остальные процессы. Это позволяет синхронизировать группы процессов.

1. **Работа без блокировок: чтение — копирование — обновление.**

Можно позволить процессу, ведущему запись, обновить структуру данных, даже если ею пользуются другие процессы. Каждый считывающий процесс читает либо старую, либо новую версию данных, но не промежуточную версию их.

Работа без блокировок структуры данных реализована потому, что **RCU** (Read – Copy - Update) отделяет фазу удаления от фазы восстановления обновления.



1. **Планирование.**

Та часть ОС, на которую возложен этот выбор, называется **планировщиком**, а алгоритм, который ею используется, называется **алгоритмом планирования**.

1. **Поведение процесса.**

Первые процессы называются процессами, **ограниченными скоростью вычислений**.

Вторые – процессами, **ограниченными скоростью работы устройств ввода-вывода**.

1. **Потребность в планирование.**

Когда планировать?

* При создании нового процесса необходимо принять решение, какой из процессов выполнять, родительский или дочерний.
* Планировщик должен принять решение, когда процесс завершает работу.
* Когда процесс блокируется в ожидании завершения операции ввода-вывода, для выполнения должен быть выбран какой-то другой процесс.
* Планировщик должен принять решение при возникновении прерывания ввода-вывода.

1. **Категории алгоритмов планирования.**

По реакции на прерывания по таймеру:

1. Неприоритетный
2. Приоритетный

Определяют три среды:

1. Пакетную
2. Интерактивную
3. Реального времени
4. **Задачи алгоритма планирования.**

Для всех систем:

* Равнодоступность – предоставление каждому процессу справедливой доли времени ЦП;
* Принуждение к определенной политике – наблюдение за выполнением установленной политики;
* Баланс – поддержка загруженности всех составных частей системы.

Для пакетных систем:

* Производительность – выполнение максимального количества заданий в час;
* Оборотное время – минимизация времени между представлением задачи и ее завершением;
* Использование ЦП – поддержка постоянной загруженности процесса.

Для интерактивных систем:

* Время отклика – быстрый ответ на запросы;
* Пропорциональность – оправдание пользовательских надежд.

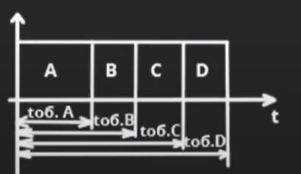
Для систем реального времени:

* Соблюдение предельных сроков – предотвращение потери данных;
* Предсказуемость – предотвращение ухудшения качества в мультимедийных системах.

1. **Алгоритм планирования в пакетных системах «Первым пришел – первым обслужен».**

Альтернативное название – FIFO.

**Задача ОС** - поддерживать некоторую очередь: поступившие на выполнение потоки добавляются в эту очередь и, допустим, по алфавиту (как ниже) пришли, выполнили и ушли.

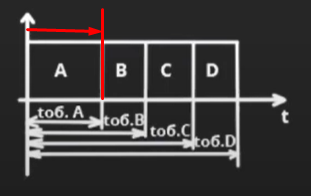


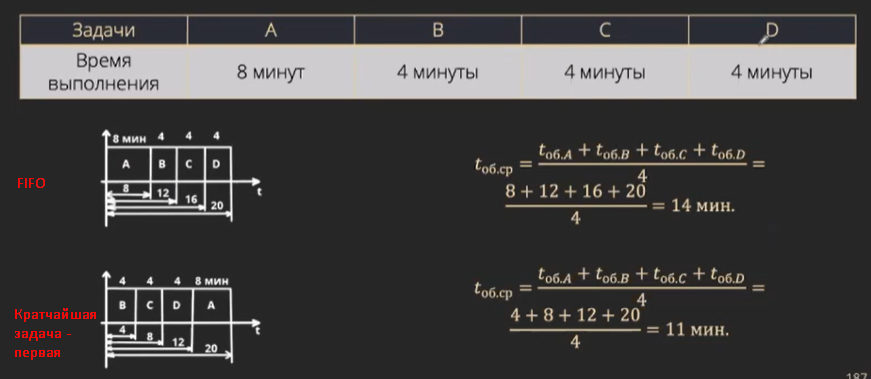
**Достоинства**:

* **Простота** - поддерживается только список, и из заголовка списка выбирается поток на выполнение.
* **Справедливость** - первым пришёл, первым будешь обслужен (как очередь в магазине).

1. **Алгоритм планирования в пакетных системах «Сначала самое короткое задание».**

Минимизирует среднее оборотное время выполнения задачи, т.е. задачи выстраиваются по критерию минимального среднего оборотного времени (чем меньше время, тем первее задача выполнится). Оборотное время - время от запуска пакета (от начала координат) до завершения каждой из задач. Для задачи А оборотное время равно от старта пакета до получения результата А, для B - время выполнения А + время выполнения B и так далее.





**Достоинства**:

* **Уменьшение оборотного времени**
* **Справедливость** - как в очереди в магазин быстрее могут пройти те, у кого без сдачи.

**Недостатки**:

* Требуется превентивная информация **о времени выполнения задач**;
* Длинный процесс, занявший процесс, **не пустит** более новые краткие процессы, пришедшие позже.

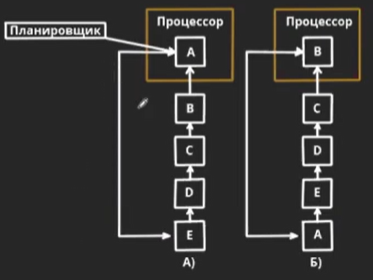
1. **Алгоритм планирования в пакетных системах «Приоритет наименьшему времени выполнения».**

В соответствии с этим алгоритмом планировщик каждый раз выбирает процесс с наименьшим оставшимся временем выполнения. В этом случае также необходимо заранее знать время выполнения задач. Когда поступает новая задача, ее полное время выполнения сравнивается с оставшимся временем выполнения текущей задачи. Если время выполнения новой задачи меньше, текущий процесс приостанавливается и управление передается новой задаче. Эта схема позволяет быстро обслуживать короткие запросы.

1. **Алгоритм планирования в интерактивных системах «Циклическое планирование».**

Альтернативное название – квантование.

Процесс выполняет поток инструкций потока А, и если квант времени истекает, то поток вытесняется, перемещается в конец очереди, и вместо него на выполнение, получив квант времени, запускается следующий в списке поток.



**Достоинства:**

* **Простота**
* **Справедливость**

**Недостатки:**

* Слишком **малый** квант времени приводит **к частому переключению** процессов и снижению производительности, но с другой стороны повышает интерактивность;
* Слишком **большой** квант может привести **к увеличению времени** **ответа** на интерактивный запрос.

1. **Алгоритм планирования в интерактивных системах «Приоритетное планирование».**

Запускается тот процесс, который находится в состоянии готовности и имеет наивысший приоритет.

Чтобы предотвратить бесконечное выполнение высокоприоритетных процессов:

* Понижать уровень приоритета текущего выполняемого процесса с каждым прерыванием по таймеру;
* Выделить каждому процессу максимальный квант допустимого времени выполнения.

Приоритеты могут присваиваться процессам в динамическом режиме.

1. **Использование нескольких очередей в интерактивных системах.**

Процессам класса с высшим приоритетом выделяется один квант, процессам следующего класса — два кванта, следующего — четыре кванта и т. д. Когда процесс использовал все отведенное ему время, он перемещается на класс ниже.

1. **Алгоритм планирования в интерактивных системах «Выбор следующего самого короткого процесса».**

Если рассматривать выполнение каждой команды как отдельную задачу, можно минимизировать общее среднее время отклика, запуская первой самую короткую задачу. Проблема состоит в том, чтобы понять, какой из ожидающих процессов самый короткий. Один из методов основывается на оценке длины процесса, базирующейся на предыдущем поведении процесса. При этом запускается процесс, у которого оцененное время самое маленькое.

1. **Алгоритм планирования в интерактивных системах «Гарантированное планирование».**

В системе с **n-процессами,** каждому **процессу**будет предоставлено 1/n времени процессора.

Необходимо отслеживать, сколько процессорного времени затрачено на каждый потоков с момента его создания. Затем вычисляют отношение времени, фактически полученного процессом к количеству времени, на которое он имел право.

На выполнение выбирается процесс с наименьшим отношением, который будет работать до тех пор, пока его соотношение не превысит соотношение его ближайшего конкурента.

1. **Алгоритм планирования в интерактивных системах «Лотерейное планирование».**

Основная идея состоит в раздаче процессам **лотерейных билетов на доступ к различным системным ресурсам**, в том числе и к процессорному времени. Когда планировщику нужно принимать решение, в случайном порядке выбирается лотерейный билет, и ресурс отдается процессу, обладающему этим билетом.

Применительно к планированию процессорного времени система может проводить лотерейный розыгрыш 50 раз в секунду, и каждый победитель будет получать в качестве приза 20 мс процессорного времени.

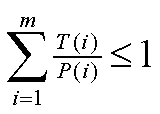
1. **Алгоритм планирования в интерактивных системах «Справедливое планирование».**

Некоторые системы перед планированием работы процесса берут в расчёт, кто является его владельцем. В этой модели каждому **пользователю, а не процессу, как в гарантированном,**распределяется некоторая доля процессорного времени и планировщик выбирает процессы, соблюдая это распределение.

Таким образом, если каждому из двух пользователей было обещано по 50% процессорного времени, то они его получат, независимо от количества имеющихся у них процессов.

1. **Планирование в системах реального времени.**

Что бы систему реального времени можно было планировать, нужно чтобы выполнялось условие:



**m** - число периодических событий

**i** - номер события

**P(i)** - период поступления события

**T(i)** - время, которое уходит на обработку события

Т.е. перегруженная система реального времени является не**планируемой**.

1. **Политика и механизмы планирования.**

1. **Планирование потоков.**

Планирование потоков, по существу, включает в себя решение двух задач:

* определение момента времени для смены текущего активного потока;
* выбор для выполнения потока из очереди готовых потоков.

1. **Модель памяти.**
2. **Адресное пространство.**

**Адресное пространство** – набор адресов, который может быть использован процессом для обращения к памяти.

**Базовый** и **ограничительный** регистры — это специальные аппаратные регистры центрального процессора, которые используются для размещения приложений в последовательно расположенных незанятых областях памяти.

В момент запуска процесса базовый регистр получает физический адрес, с которого размещается в памяти запущенная программа, а ограничительный регистр — значение длины программы.

1. **Свопинг.**

Суть **свопинга** заключается в размещении в памяти всего процесса целиком, его запуске на некоторое время, а затем сбросе на диск.

Когда в результате свопинга в памяти создаются несколько свободных областей, их можно объединить в одну большую за счет перемещения всех процессов в нижние адреса. Эта технология известна как **уплотнение памяти**.

1. **Распределение памяти в динамическом режиме.**

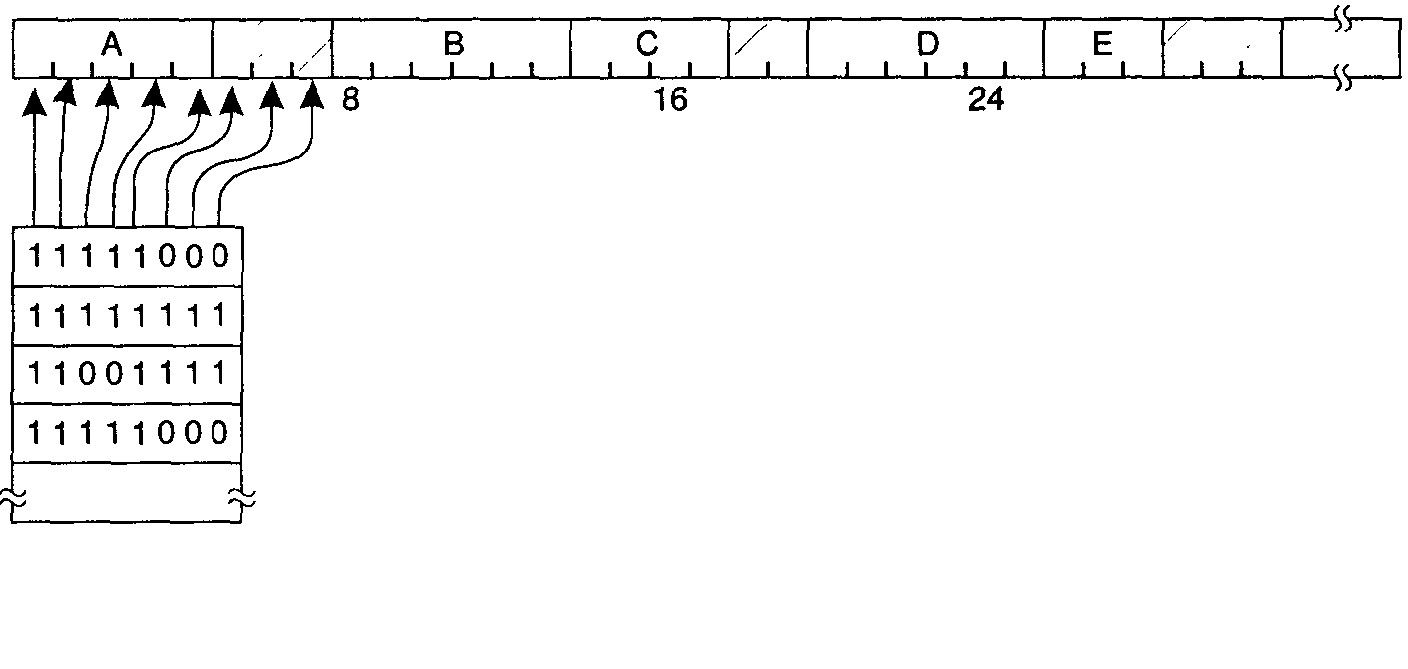
Та часть ОС, которая управляет иерархией памяти (или ее частью), называется менеджером, или диспетчером памяти.

Если память распределяется в динамическом режиме, то управлять этим должна ОС.

Существуют два способа отслеживания использования памяти:

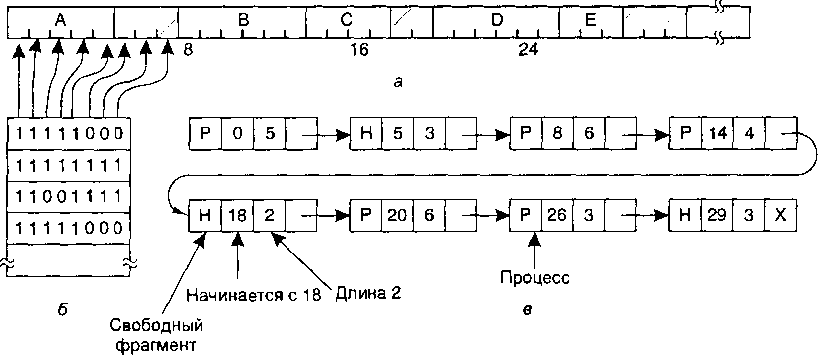
1. Битовые матрицы
2. Списки свободного пространства.
3. **Управление памятью с помощью битовых матриц.**

При использовании битовых матриц память делится на единичные блоки размером он нескольких слов до нескольких килобайт. С каждым единичным блоком соотносится один бит в битовой матрице, который содержит 0, если единичный блок свободен, и 1, если он занят (или наоборот).



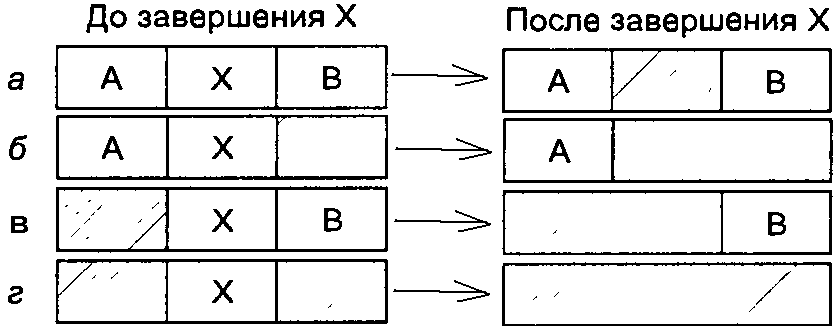
1. **Управление памятью с помощью связанных списков.**

Каждая запись в списке хранит обозначения: сегмент “Дыру” – hole (H) или процесс – process (P), адрес, с которого сегмент начинается, его длину и указатель на следующую запись.



1. **Сортировка списка сегментов по адресам.**

Сортировка списка сегментов по адресам дает преимущество, что при завершении процесса или его свопинге на диск упрощается обновление списка.



1. **Алгоритм выделения памяти процессу – «Перове подходящее».**

* Диспетчер памяти сканирует список сегментов.
* Затем пустое пространство разбивается на две части.

“Первое подходящее” – это быстрый алгоритм, поскольку поиск ведется с наименьшими затратами времени.

1. **Алгоритм выделения памяти процессу – «Следующее подходящее».**

* Диспетчер памяти сканирует список сегментов.
* Отслеживает свое местоположение.
* Затем пустое пространство разбивается на две части.

Его производительность несколько хуже, чем алгоритма “первое подходящее”.

1. **Алгоритм выделения памяти процессу – «Наиболее подходящее».**

* Диспетчер памяти сканирует список сегментов.
* Выбирается наименьшее соответствующее пустое пространство.

Работает медленнее, чем “первое подходящее”, поскольку при каждом вызове он должен вести поиск по всему списку.

1. **Алгоритм выделения памяти процессу – «Наименее подходящее».**

* Диспетчер памяти сканирует список сегментов.
* Выбирается самого большого подходящего пустого пространства.
* Затем пустое пространство разбивается на две части.

Работает медленнее, чем “первое подходящее”, поскольку при каждом вызове он должен вести поиск по всему списку.

1. **Виртуальная память.**

В 1960-е годы было принято решение разбивать программы на небольшие части, называемые **оверлеями**.

При запуске программы в память загружался только администратор оверлейной загрузки, который тут же загружал и запускал оверлей с порядковым номером 0.

Изобретенный метод стал известен как **виртуальная память**. У каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые **страницами**.

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, ОС предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду.

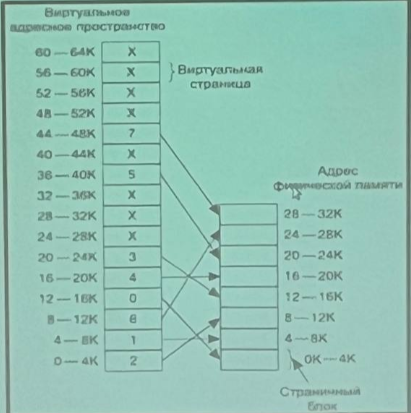
1. **Страничная организация памяти.**

**Страничная организация памяти** — способ организации виртуальной памяти, при котором виртуальные адреса отображаются на физические постранично.

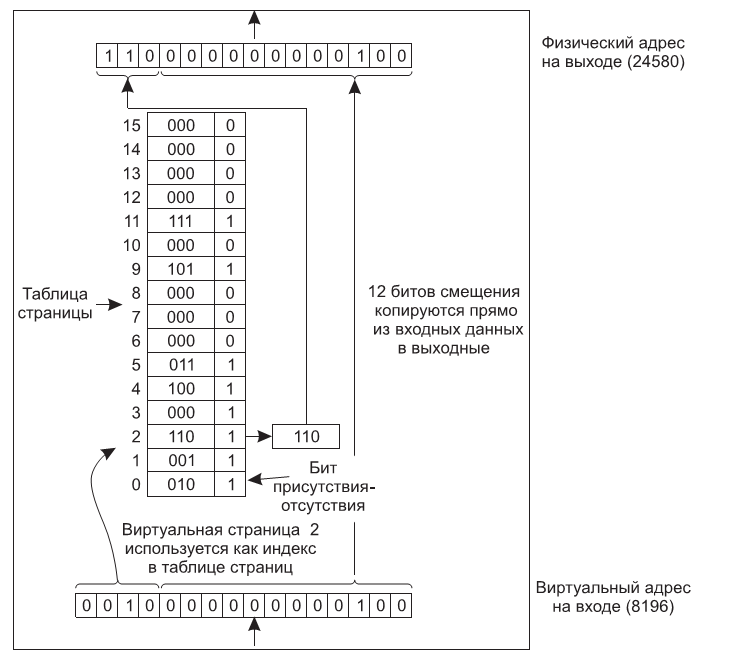
1. **Связь виртуальной и физической памяти.**

Виртуальное адресное пространство состоит из блоков фиксированного размера, называемых **страницами**. Соответствующие блоки в физической памяти называются **страничными блоками**.

Страницы и страничные блоки имеют, как правило, одинаковые размеры.



1. **Внутреннее устройство диспетчера памяти.**



1. **Структура записи в таблице страниц.**

* Поле номера страничного блока;
* Бит присутствия-отсутствия;
* Биты защиты сообщают о том, какого рода доступ разрешен, со значением 0 для чтения-записи и значением 1 только для чтения;
* Бит модификации, когда в страницу осуществляется запись, аппаратура автоматически устанавливает его;
* Бит ссылки устанавливается при обращении к страницу, помогает ОС выбрать выселяемую страницу при возникновении ошибки отсутствия страницы;
* Бит блокирование кэширования страницы, аппаратура продолжает извлечение слова из устройства, а не использует старую копию, попавшую в кэш.

1. **Ускорение работы страничной организации памяти.**

* Скорость отображения виртуального адреса на физический.

Отображение виртуальной памяти на физическую должно осуществляться при каждом обращении к памяти. Все команды должны поступать из памяти, и многие из них ссылаются на операнды, которые также находятся в памяти.

* Большой размер таблицы страниц, если пространство виртуальных адресов слишком обширное.

При 1 млн страниц в виртуальном адресном пространстве таблица страниц должна содержать 1 млн записей. Также каждому процессу требуется собственная таблица страниц (поскольку у него собственное виртуальное адресное пространство).

* Буферы быстрого преобразования адреса.
* Программное управление буфером TLB.

1. **Буферы быстрого преобразования адреса (TLB).**

**Буфер быстрого преобразования адреса (TLB)** – устройство для отображения виртуальных адресов на физические без просмотра таблицы страниц.

1. **Программное управление буфером TLB.**

На ПК записи TLB загружаются ОС явным образом. Когда нужная запись в TLB отсутствует, диспетчер памяти, вместо того, чтобы обращаться к таблицам страниц для поиска и извлечения сведений о нужной странице, просто генерирует TLB-ошибку и подбрасывает проблему ОС.

1. **Страничная организация памяти - Ошибки отсутствия записей.**

**Программная ошибка отсутствия** происходит, когда страница, к которой идет обращение, отсутствует в TLB, но присутствует в памяти.

**Аппаратная ошибка отсутствия** происходит, когда сама страница отсутствует в памяти (и запись о ней отсутствует в TLB).

Аппаратная обрабатывается медленнее, чем программная. Просмотр отображения в иерархии таблиц страниц называется просмотром таблиц страниц.

1. **Многоуровневые таблицы страниц.**

Таблицы страниц для больших объемов памяти.

Суть метода заключается в отказе от постоянного хранения всех таблиц страниц в памяти. В частности, не должны храниться те таблицы, в которых нет необходимости.

1. **Инвертирование таблицы страниц.**

Если каждая запись будет занимать 8 байт, то размер таблицы превысит 30 миллионов байт.

Решение данной проблемы является использование **инвертированной таблицы страниц**. В этой конструкции одна запись выделяется не каждой странице виртуального адресного пространства, а каждому страничному блок в реальной памяти.

Инвертированные таблицы страниц имеют серьезный недостаток: перевод виртуального адреса в физический становится намного сложнее. Когда процесс N обращается к виртуальной странице p, аппаратное обеспечение не может больше найти физическую страницу, используя номер p в качестве индекса в таблице страниц. Вместо этого оно должно производить поиск записи (n, p) во всей инвертированной таблице страниц.

1. **Hash-таблица.**

Все виртуальные страницы, находящиеся в данный момент в памяти и имеющие одинаковое значение хэш-функции, сцепляются друг с другом. Если хэш-таблица состоит из такого же количества ячеек, сколько в машине физических страниц, средняя цепочка будет длиной только в одну запись, что значительно увеличит скорость отображения адресов. Как только найден номер страничного блока, новая пара помещается в буфер TLB.

1. **Алгоритмы замещения страниц.**

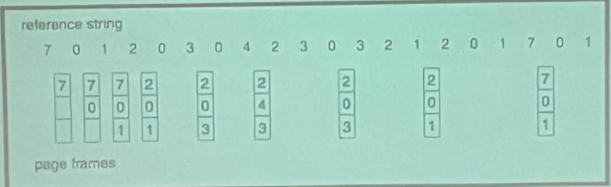
Когда происходит страничное прерывание, ОС должна выбрать страницу для удаления из памяти, чтобы освободить место для страницы, которую нужно перенести в память. Если удаляемая страница была изменена за время своего присутствия в памяти, ее необходимо переписать на диск, чтобы обновить копию, хранящуюся там. Однако если страница не была модифицирована, копия на диске уже является самой новой и ее не надо переписывать. Тогда страница, которую нужно прочитать, просто считывается поверх выгружаемой страницы.

* Оптимальный алгоритм
* Алгоритм NRU – не использовавшаяся в последнее время страница
* Алгоритм FIFO – первым прибыл – первым обслужен
* Алгоритм “Вторая попытка”
* Алгоритм “Часы”
* Алгоритм LRU – страница, не использовавшаяся дольше всего
* Программное моделирование алгоритма LRU
* Алгоритм “Рабочий набор”
* Алгоритм WSClock

1. **Оптимальный алгоритм замещения страниц.**

**Цель данного алгоритма** – выбрать страницу с максимальным временем, т.е. обращение к которой будет позже других.

**Оптимальный** **алгоритм** **замещения** **страниц** гласит, что должна быть удалена **страница**, имеющая пометку с наибольшим значением. Если какая-то **страница** не будет использоваться на протяжении 8 миллионов команд, а другая какая-нибудь **страница** не будет использоваться на протяжении 6 миллионов команд, то удаление первой из них приведет к ошибке отсутствия **страницы**, в результате которой она будет снова выбрана с диска в самом отдаленном будущем.



1. **Алгоритм исключения недавно использовавшейся страницы.**

У страничного блока есть два специальных бита: обращения (R) и изменения (M), который устанавливаются при чтении и записи соответственно.

При запуске процесса оба страничных бита для всех его страниц устанавливается в 0. Время от времени бит R сбрасывается, чтобы отличить те страницы, к которым в последнее время не было обращений, от тех, к которым такие обращения были.

При возникновении ошибки отсутствия страницы ОС просматривает все страницы и делит из на 4 категории.

1. В последнее время не было ни обращений, ни модификаций.
2. Обращений в последнее время не было, но страница модифицирована.
3. В последнее время были обращения, но модификация не было.
4. В последнее время были и обращения, и модификации.

Алгоритм исключения недавно использовавшейся страницы (NRU) удаляет произвольную страницу, относящуюся к самой низкой непустой категории.

1. **Алгоритм «Первой пришла, первой ушла».**

При использовании метода FIFO ОС ведет список всех страниц, находящихся на данный момент в памяти, причем совсем недавно поступившие находятся в хвосте, поступившие раньше всех – в голове списка. При возникновении ошибки отсутствия страницы удаляется страница, находящаяся в голове списка, а к его хвосту добавляется новая страница.

Главный недостаток – наиболее старая страница не всегда будет являться наименее используемой. Поэтому принцип FIFO в его чистом виде используется довольно редко.

1. **Алгоритм «Второй шанс».**

Алгоритм «второй шанс» — это модификация алгоритма FIFO, которая исключает проблему удаления часто востребованной страницы.

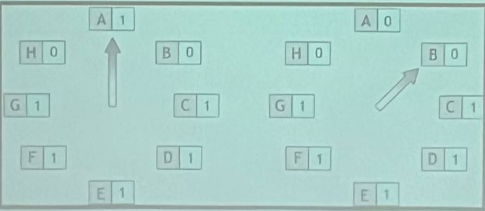
Он заключается в проверке бита R самой старой страницы:

1. Если его значение равно нулю, значит, страница не только старая, но и невостребованная, поэтому она тут же удаляется.
2. Если бит R имеет значение 1, он сбрасывается, а страница помещается в конец списка страниц и время ее загрузки обновляется, как будто она только что поступила в память.
3. Затем поиск продолжается.

Алгоритм «второй шанс» занимается поиском ранее загруженной в память страницы, к которой за только что прошедший интервал времени таймера не было обращений.

1. **Алгоритм «Часы».**

При возникновении ошибки отсутствия страницы проверяется та страница, на которой находится указатель (стрелка). Если ее бит R имеет значение 0, страница замещается, и указатель передвигается вперед на одну позицию. Если значение бита R равно 1, то он сбрасывается, и указатель перемещается на следующую страницу. Процесс повторяется до тех пор, пока не будет найдена страница с R равным 0.



1. **Алгоритм замещения наименее востребованной страницы.**

Идея заключается в том, что при возникновении ошибки отсутствия страницы нужно избавиться от той страницы, которое длительное время была не востребована.

Для его полной реализации необходимо вести связанный список всех страниц, находящихся в памяти. В начале этого списка должна быть только что востребованная страница, а в конце – наименее востребованная. Сложность в том, что этот список должен обновляться при каждом обращении к памяти. Для поиска страницы в списке, ее удаления из него и последующего перемещения этой страницы вперед потребуется довольно много времени, даже если это будет возложено на аппаратное обеспечение.

1. **Основные требования к долговременному хранилищу данных.**
2. Должно предоставляться возможность хранения огромного количества информации.
3. Информация должна пережить прекращение работы использующего ее процесса.
4. К информации должны иметь одновременный доступ несколько процессов.
5. **Имена файлов.**

Когда процесс создает файл, он присваивает ему имя. Когда процесс завершается, файл продолжает существовать, и к нему по этому имени могут обращаться другие процессы.

В именование файлов допускается применение цифр и специальных символов. Многие файловые системы поддерживают имена длиной до 255 символов.

Некоторые файловые системы различают буквы верхнего и нижнего регистров (например, UNIX), а некоторые нет (MS-DOS).

**Расширение имени файла** – информация о файле, которая следует за точкой в имени.

1. **Структура файла.**
2. **Последовательность байтов**. ОС не интересуется содержимым файла, она видит только байты. Используется в Windows и UNIX.
3. **Последовательность записей**. Записи фиксированной длины (например, перфокарта) считываются последовательно. Сейчас не используются.
4. **Дерево записей**. Каждая запись имеет ключ, записи считываются по ключу. Используется на мэйнфреймах.



1. **Типы файлов.**

* **Обычные** файлы, содержащие информацию пользователя. Это либо файлы ASCII, либо двоичные.
* **Каталоги** – системные файлы, предназначенные для поддержки структуры файловой системы.
* **Символьные специальные файлы** имеют отношения к вводу-выводу и используются для моделирования последовательных устройств ввода-вывода, к которым относятся терминалы, принтеры и сети.
* **Блочные специальные файлы** используются для моделирования дисков.

1. **Доступ к файлам.**

В самых первых ОС предоставлялся только один тип доступа к файлам – **последовательный**.

Файлы, в которых байты или записи могли быть считаны в любом порядке, стали называть **файлами произвольного доступа**.

Для определения места начала считывания могут быть применены два метода. При первом методе позиция в файле, с которой начинается чтение, задается при каждой операции чтения read.

При втором методе для установки на текущую позицию предоставляется специальная операция поиска нужного места seek. После этой операции файл может быть считан последовательно с только что установленной позиции. Последний метод используется в UNIX и Windows.

1. **Атрибуты файлов.**

У каждого файла есть свои имя и данные. Вдобавок к этому все ОС связывают с каждым файлом и другую информацию, к примеру, дату и время последней модификации файла и его размер.

Эти доп. Сведения называются **атрибутами файла**.

Вот некоторые из них: Защита, Пароль, Создатель, Владелец, Флаг “только для чтения”, Флаг “скрытый”.

1. **Системы с одноуровневыми каталогами.**

Обычно в файловой системе для упорядочения файлов имеются каталоги или папки, которые сами по себе являются файлами.

Самая простая форма системы каталогов состоит из одного каталога, содержащего все файлы. Иногда он называется **корневым каталогом**, но поскольку он единственный, то имя особого значения не имеет.

Преимущество такой схемы заключается в ее простоте и возможности быстрого нахождения файлов, поскольку поиск ведется всего в одном месте.

1. **Иерархические системы каталогов.**

Для организации файлов нужна иерархия. Такой подход позволяет иметь столько каталогов, сколько необходимо для группировки файлов естественным образом.

Кроме того, если файловый сервер используется несколькими пользователями, то каждый пользователь может иметь свой собственный каталог.

1. **Структура файловой системы.**

Файловые системы хранятся на дисках. Большинство дисков может быть разбито на несколько разделов, на каждом из которых будет независимая файловая система. Сектор 0 на диске называется **главной загрузочной записью** **(MBR)** и используется для загрузки компьютера.

В конце MBR содержится таблица разделов. Из этой таблицы берутся начальные и конечные адреса каждого раздела. Один из разделов в этой таблице помечается как активный.

При загрузке компьютера BIOS считывает и выполняет MBR. Первое, что делает программа MBR, - находит расположение активного раздела, считывает его первый блок, который называется загрузочным, и выполняет его. Программа в загрузочном блоке загружает ОС, содержащуюся в этом разделе.

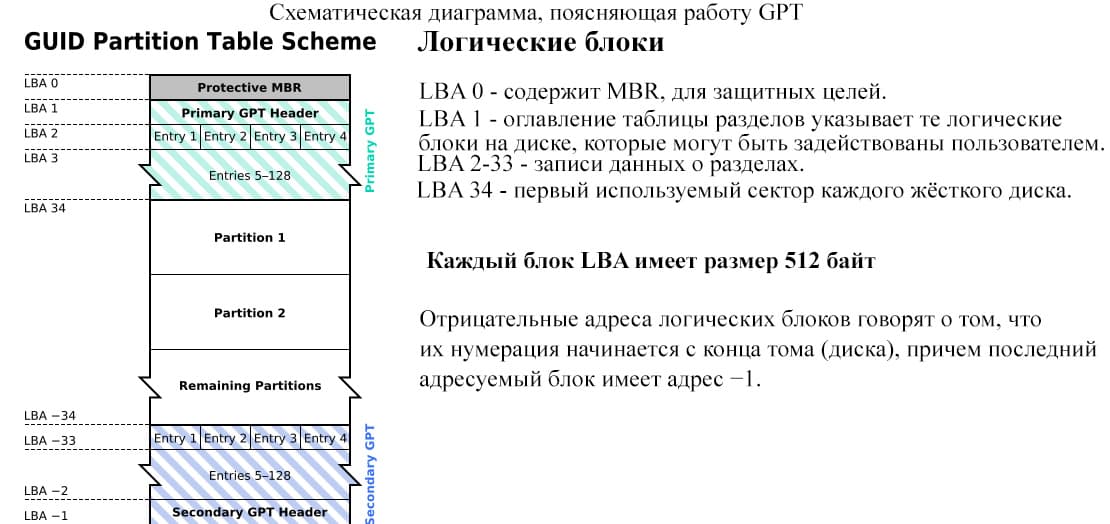
Первым элементом является **суперблок**. В нем содержатся все ключевые параметры файловой системы, которые считываются в память при загрузке компьютера или при первом обращении к файловой системе.  
Далее может находиться информация о свободных блоках файловой системы, к примеру в виде битового массива или списка указателей. За ней могут следовать i-узлы, массив структур данных. Затем размещается корневой каталог, содержащий вершину дерева файловой системы. И наконец, оставшаяся часть диска содержит все остальные каталоги и файлы.

1. **GUID-таблица разделов.**

**GUID-таблица разделов (GPT)** была разработана в связи с необходимостью преодоления ограничений, накладываемых традиционной таблицей разделов жесткого диска, не позволяющей создавать разделы размером свыше 2 Тбайт или располагать их на дисках за пределами первых 2 Тбайт их емкости.

Важнейшим техническим отличием GPT от традиционной таблицы разделов является полный отказ от использования дисковых адресов формата CHS и окончательный переход на адресацию LBA.

В отличии от адресации CHS, LBA никак не связана с физическими характеристиками диска: весь диск в этой системе считается непрерывным массивом блоков одинакового размера, причем их нумерация ведется с нуля.



1. **Примеры файловых систем для Unix, Windows и MacOS.**

Unix: JFS, SWAPFS, XFSC, ReiserFS, Ext2/3

Windows: NTGS, FAT32, FAT16, FAT12

MacOS: HFS, HFS+

1. **Устройства ввода-вывода.**

Устройства ввода-вывода можно разделить на 2 категории: **блочные** и **символьные** устройства.

К **блочным** относятся такие устройства, которые хранят информацию в блоках фиксированной длины, у каждого из которых есть собственный адрес. Важным свойством блочного устройства является то, что оно способно читать или записывать каждый блок независимо от всех других блоков. Среди наиболее распространенных блочных устройств жесткие диски, приводы Blu-ray-дисков и флеш-накопители USB.

**Символьные** устройства выдают или воспринимают поток символов, не относящийся ни к какой блочной структуре. Они не являются адресуемыми и не имеют никакой операции позиционирования. В качестве символьных устройств могут рассматриваться принтеры, сетевые интерфейсы, мыши и т.д.

1. **Назначение подсистемы ввода-вывода.**

Назначение - обмен данными между пользователями, приложениями и периферийными устройствами компьютера.

1. **Задачи подсистемы ввода-вывода.**

* Организация параллельной работы устройств ввода-вывода и процессора
* Согласование скоростей обмена и кэширование данных
* Разделение устройств и данных между процессами
* Обеспечение удобного логического интерфейса между устройствами и остальной частью системы
* Поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера
* Динамическая загрузка и выгрузка драйверов без дополнительных действий с ОС
* Поддержка нескольких различных файловых систем
* Поддержка синхронных и асинхронных операций ввода-вывода

1. **Технологии подсистемы ввода-вывода.**

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами.

1. **С помощью программируемого ввода-вывода**. В этом случае, когда процессору встречается команда, связанная с вводом-выводом, он выполняет ее, посылая соответствующие команды контроллеру ввода-вывода. Это устройство выполняет требуемое действие, а затем устанавливает соответствующие биты в регистрах состояния ввода-вывода и не посылает никаких сигналов, в том числе сигналов прерываний. Процессор периодически проверяет состояние модуля ввода-вывода с целью проверки завершения операции ввода-вывода.

Таким образом, процессор непосредственно управляет операциями ввода-вывода, включая опознание состояния устройства, пересылку команд чтения-записи и передачу данных. Процессор посылает необходимые команды контроллеру ввода-вывода и переводит текущий процесс в состояние ожидания завершения операции ввода-вывода. Недостатки такого метода – большие потери процессорного времени, связанные с управлением вводом-выводом.

1. **Ввод-вывод, управляемый прерываниями**. Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять текущий процесс, если нет необходимости в ожидании выполнения операции ввода-вывода. В противном случае текущий процесс приостанавливается до получения *сигнала прерывания* о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд.

Такой ввод-вывод намного эффективнее, чем программируемый ввод-вывод, так как при этом исключается ненужное ожидание с бесполезным простоем процессора. Однако и в этом случае ввод-вывод потребляет еще значительное количество процессорного времени, потому что каждое слово, которое передается из памяти в модуль ввода-вывода (контроллер) или обратно, должно пройти через процессор.

1. **Прямой доступ к памяти** (**DMA**). В этом случае специальный модуль прямого доступа к памяти управляет обменом данных между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных.

В настоящее время в персональных и других компьютерах используется третий способ ввода-вывода, поскольку в структуре компьютера имеется DMA-контроллер или подобное ему устройство, обслуживающее, как правило, запросы по передаче данных от нескольких устройств ввода-вывода на конкурентной основе.

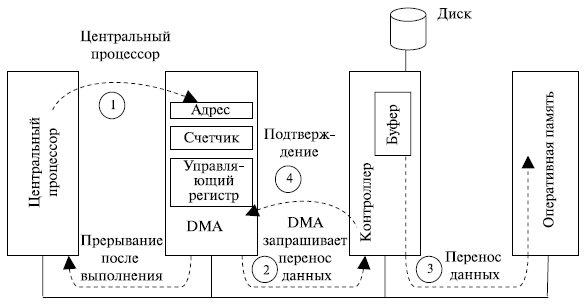
1. **DMA-контроллер.**

DMA (Direct Memory Access) - специализированный контроллер, предназначенный для пересылки данных без использования ядра микроконтроллера.

Использование DMA позволяет не тратить процессорное время на отслеживание готовности данных в периферийных блоках и последующую их пересылку. Ядро инициализирует DMA и переходит к выполнению других задач, пока не возникнет прерывание от DMA, сигнализирующее о том, что заданное количество данных было передано.

1. **Прямой доступ к памяти (Direct Memory Access (DMA)).**

**Прямой** **доступ** **к** **памяти** (англ. **direct** **memory** **access**, **DMA**) — режим обмена данными между устройствами компьютера или же между устройством и основной **памятью**, в котором центральный процессор (ЦП) не участвует. Так как данные не пересылаются в ЦП и обратно, скорость передачи увеличивается.



1. **Прерывания.**

Прерывания — это система, которая переводит процессор на выполнение потока команд, отличного от того, который выполнялся до сих пор, с последующим возвратом к исходному коду.

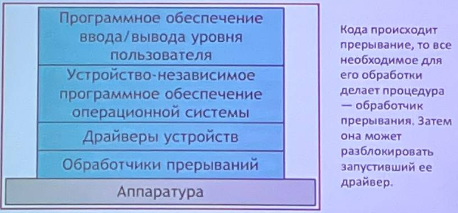


1. **Программное обеспечение ввода/вывода.**

Основные задачи, которые должно решать ПО ввода-вывода:

* Независимость от устройств
* Единообразное именование
* Обработка ошибок
* Перенос данных
* Буферизация
* Проблема выделенных и невыделенных устройств

1. **Уровни программного обеспечения.**



1. **Обработчики прерываний.**

Прерывания должны быть скрыты как можно глубже в недрах ОС, чтобы как можно меньшая часть ОС имела с ними дело. Лучше всего блокировать драйвер, начавший ввод-вывод.

Алгоритм:

1. Драйвер начинает операцию ввод-вывод
2. Драйвер блокирует сам себя,
   * Выполнив на семафоре процедуру down
   * Выполнив на переменной состояния процедуру wait
   * Выполнив на сообщении процедуру receive
3. Происходит прерывание
4. Обработчик прерываний начинает работу
5. Обработчик прерываний может разблокировать драйвер
6. **Драйверы устройств.**

**Драйвер** — это компьютерное программное обеспечение, с помощью которого другое программное обеспечение (операционная система) получает доступ к аппаратному обеспечению некоторого устройства.

Обмен данными между контроллерами и драйверами идет по шине. Драйвера должны взаимодействовать с ОС через стандартные интерфейсы.

Стандартные интерфейсы, которые должны поддерживать драйвера:

* Для блочных устройств
* Для символьных устройств

Раньше для установки ядра приходилось перекомпилировать ядра системы.

Сейчас в основном ОС загружают драйверы. Некоторые могут быть загружены в горячем режиме.

Функции, которые выполняют драйвера:

* Обработка запросов чтения или записи
* Инициализация устройства
* Управление энергопотреблением устройства
* Прогрев устройства
* Включение устройства или запуска двигателя

1. **Программное обеспечение ввода-вывода, не зависящее от конкретных устройств.**

Функции независимого от устройств ПО ввода-вывода:

* Единообразный интерфейс для драйверов устройств

Кроме интерфейса, в него также входят проблемы:

* + Именование устройств
  + Защита устройств
* Буферизация
* Сообщения об ошибках
* Захват и освобождение выделенных устройств (блокирование)
* Предоставление унифицированного размера блока, не зависящего от конкретных устройств.

1. **Буферизация.**

**Буферизация** — метод организации обмена, в частности, ввода и вывода данных в компьютерах и других вычислительных устройствах, который подразумевает использование буфера для временного хранения данных

1. Небуферизованный ввод – после ввода каждого символа происходит прерывание
2. Буферизация в пространстве пользователя – приходится держать загруженными необходимые страницы памяти в физической памяти
3. Буферизация в ядре с копированием в пространство пользователя – страница загружается только когда буфер ядра полный, данные из буфера ядра в буфер пользователя копируется за одну операцию. Проблема может возникнуть, когда буфер полный, а страница буфера пользователя еще не загружена.
4. Двойная буферизация в ядре – если один буфер заполнен, и пока он выгружается, символы пишутся во второй буфер.
5. **Взаимоблокировка.**

**Взаимоблокировка** – ситуация, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими процессами.

1. **Выгружаемые и невыгружаемые ресурсы.**

**Выгружаемый ресурс**- этот ресурс безболезненно можно забрать у процесса (например: память).

**Невыгружаемый ресурс -**этот ресурс нельзя забрать у процесса без потери данных (например: принтер).

Проблема взаимоблокировок процессов возникает при борьбе за невыгружаемые ресурсы.

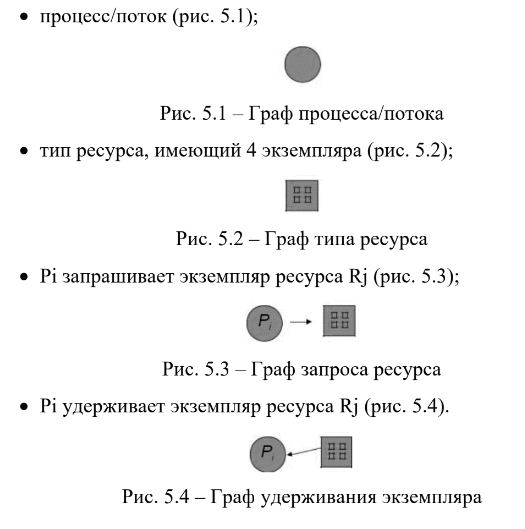
1. **Последовательность событий при использовании ресурса.**
2. Запрос ресурса.
3. Использование ресурса.
4. Высвобождение ресурса.
5. **Условия возникновения ресурсных взаимоблокировок.**

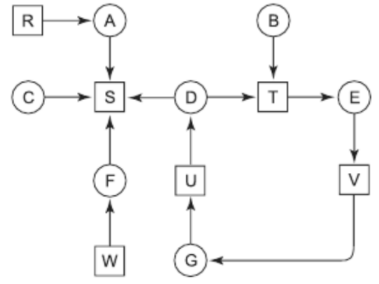
Условия Коффмана:

1. Условие взаимного исключения. Каждый ресурс либо выделен в данный момент только одному процессу, либо доступен.
2. Условие удержания и ожидания. Процессы, удерживающие в данный момент ранее выделенные им ресурсы, могут запрашивать новые ресурсы.
3. Условие невыгружаемости. Ранее выделенные ресурсы не могут быть принудительно отобраны у процесса. Они должны быть явным образом высвобождены тем процессом, который их удерживает.
4. Условие циклического ожидания. Должна существовать кольцевая последовательность из двух и более процессов, каждый из которых ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.

Для возникновения ресурсной взаимоблокировки должны соблюдаться все четыре условия.

1. **Ресурсный граф.**





1. **Действие алгоритма обнаружения взаимоблокировки при использовании одного ресурса каждого типа.**
2. Для каждого узла N, имеющегося в графе, выполняются следующие пять шагов, использующих узел N в качестве начального.
3. Инициализируется список L (динамическая структура данных, представляющая собой список узлов, а также список ребер), а со всех ребер снимаются пометки.
4. Текущий узел добавляется к концу списка L, и проводится проверка, не появится ли этот узел в списке L дважды. Если это произойдет, значит, граф содержит цикл (отображенный в списке L), и алгоритм прекращает работу.
5. Для заданного узла определяется, нет ли каких-нибудь отходящих от него непомеченных ребер. Если такие ребра есть, осуществляется переход к шагу 5, если их нет, то к шагу 6.
6. Произвольно выбирается и помечается непомеченное отходящее от узла ребро. Затем по нему осуществляется переход к новому текущему узлу, и алгоритм возвращается к шагу 3.
7. Если этот узел является первоначальным узлом, значит, граф не содержит никаких циклов, и алгоритм завершает свою работу. В противном случае, алгоритм зашел в тупик. Этот узел удаляется, и алгоритм возвращается к предыдущему узлу. Данный узел делается текущим, и осуществляется переход к шагу 3.
8. **Выход из взаимоблокировки.**

* **Восстановление за счет приоритетного овладения ресурсом.** Отбирается ресурс у текущего владельца и передается другому.
* **Восстановление путем отката.** Периодически процессы создают контрольные точки, к которым могут вернуться при взаимоблокировки.
* **Восстановление путем уничтожения процессов.** Уничтожается процесс, находящийся в цикле взаимоблокировки.

1. **Уклонение от взаимоблокировок.**

Для того чтобы избежать состояния гонки, необходимо следовать определенным рекомендациям и правилам. В частности, нужно избегать общих ресурсов, использовать потокобезопасные коллекции и другие структуры данных, минимизировать блокировки и синхронизацию, избегать глобальных переменных и других общих ресурсов, использовать атомарные операции, когда это возможно.

1. **Предотвращение взаимоблокировок.**

* Использование правильной последовательности блокировки
* Использование объектов Lock.

1. **Многопроцессорные системы.**
2. **Увеличение производительности ЦП.**

**Закон Мура** – По мере совершенствования технологии производства микросхем транзисторы становились все меньше и меньше, и появилась возможность размещать их на микросхемах все в большем и большем количестве.

Другая возможность заключается в размещении на одной и той же микросхеме двух и более полноценных ЦП, обычно называемых **ядрами**.

1. **Внешние условия, требующие принятия дополнительных мер безопасности.**
2. **Угрозы.**

Безопасность информационных систем разбита на три компонента: конфиденциальность, целостность и доступность. Вместе они называются CIA (Confidentiality, Integrity, Availability).

* **Конфиденциальность** – направлено на сохранение секретности данных.
* **Целостность** – означает, что пользователи, не обладающие необходимыми правами, не должны иметь возможности изменять какие-либо данные без разрешения их владельцев.
* **Доступность** – означает, что никто не может нарушить работу системы и вывести ее из строя.
* **Взломщики** – люди, пытающиеся взламывать не принадлежащие им компьютерные системы.
* Некоторые средства помогают взломщикам посредством **сканирования портов** выявлять сетевые услуги, предлагаемые компьютерной системой. Например, попытка TCP-подключений к каждому возможному номеру порта на компьютерной системе.
* Основная часть преступных действий в Интернете построена на **бот-сетях**, которые состоят из тысяч (а иногда и миллионов) зараженных компьютеров, зачастую являющихся обычными компьютерами ничего не подозревающих об этом пользователей.
* Еще одним аспектом безопасности, имеющим отношение к конфиденциальности, является **закрытость**: защита отдельных пользователей от злоупотреблений, связанных с их личной информацией.

1. **Злоумышленники.**

**Злоумышленник** – человек, намеренно обходящий компьютерную безопасность.

Попытка реализации угрозы называется **атакой**, а тот, кто предпринимает такую попытку - **злоумышленником**. Потенциальные злоумышленники называются **источниками угрозы**.

1. **Управление доступом к ресурсам.**

**Домен** представляет собой множество пар (объект, права доступа). Каждая пара определяет объект и некоторое подмножество операций, которые могут быть выполнены в отношении этого объекта.

**Права доступа** означают в данном контексте разрешение на выполнение той или иной операции. Домен соотносится с отдельным пользователем или группой пользователей, сообщая о том, что может, а что не может сделать этот пользователь.

* Списки управления доступом (ACL).
* Перечни возможностей. При использовании этого метода с каждым процессом будет связан список объектов, к которым может быть получен доступ, а также информация об операциях с ними. Такой список называется **перечнем возможностей**, а его элементы – **возможностями**.

1. **Стадии защиты информации.**
2. **Идентификация.** Распознавание информации о пользователе, например, логин и пароль.
3. **Аутентификация.** Процесс проверки информации о пользователе.
4. **Авторизация.** Проверка прав пользователя и определение возможности доступа.



1. **Аутентификация.**

**Аутентификация** – основа безопасности любой системы, которая заключается в проверке подлинности данных о пользователе сервером.

Элементы аутентификации:

* **Субъект** – пользователь.
* **Характеристика субъекта** – информация, предоставляемая пользователем для проверки подлинности.
* **Владелец системы аутентификации** – владелец ресурса.
* **Механизм аутентификации** – принцип проверки.
* **Механизм авторизации** – управление доступом.

Методы аутентификации:

* **Парольные.** С помощью пароля.
* **Комбинированные.** Используются несколько методов.
* **Биометрические.** Например, по отпечатку пальца.
* **Информация о пользователе.** Номер телефона, год рождения и т.д.
* **Пользовательские данные.** Основан на геоданных о местоположении пользователя с использованием GPS.

1. **Классификация видов аутентификации.**

**В зависимости от количества используемых методов:**

* **Однофакторная.** Используется только один метод.
* **Многофакторная.** Используются несколько методов.

**В зависимости от политики безопасности систем и уровня доверия:**

* **Односторонняя**. Пользователь доказывает право доступа к ресурсу его владельцу
* **Взаимная**. Проверяется подлинность прав доступа и пользователя и владельца сайта. Для этого используют криптографические способы.

1. **Средства защиты.**

**Средства защиты информации** — это совокупность инженерно-технических, электрических, электронных, оптических и других устройств и приспособлений, приборов и технических систем, а также иных вещных элементов, используемых для решения различных задач по защите информации, в том числе предупреждения утечки и обеспечения безопасности защищаемой информации.

Средства защиты информации можно разделить на следующие группы:

1. **Технические (аппаратные) средства**: различные по типу устройства (механические, электромеханические, электронные и др.), которые аппаратными средствами решают задачи защиты информации.
2. **Программные средства**: включают программы для идентификации пользователей, контроля доступа, шифрования информации, удаления остаточной (рабочей) информации типа временных файлов, тестового контроля системы защиты и др.
3. **Смешанные аппаратно-программные средства**: реализуют те же функции, что аппаратные и программные средства в отдельности, и имеют промежуточные свойства.
4. **Организационные средства**: складываются из организационно-технических и организационно-правовых средств