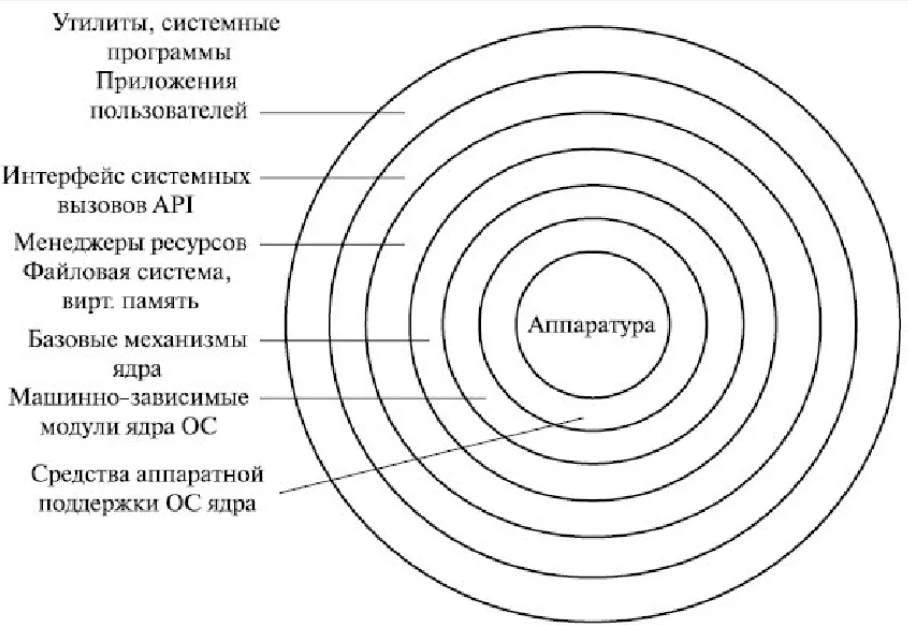
1. **Изобразить структуру операционной системы. Объяснить назначение компонентов.**



В данной схеме выделены следующие слои.

Средства ***аппаратной*** поддержки ОС. Значительная часть функций ОС может выполняться аппаратными средствами [13]. Чисто программных ОС сейчас не существу-ет. Как правило, в современных системах всегда есть средства аппаратной поддержки ОС, которые прямо участвуют в организации вычислительного процесса. К ним относятся: система прерываний, средство поддержки привилегированного режима, средства поддержки виртуальной памяти, системный таймер, средство переключения контекстов процессов (информация о состоянии процесса в момент его приостановки), средство защиты памяти и др.

Машинно-зависимые модули ОС. Этот слой образует модули, в которых отражается специфика аппаратной платформы компьютера. Назначение этого слоя – "экранирование" вышележащих слоев ОС от особенностей аппаратуры (например, Windows 2000 – это слой HAL, Hardware Abstraction Layer, уровень аппаратных абстракций).

Базовые механизмы ядра. Этот слой модулей выполняет наиболее примитивные операции ядра: программное переключение контекстов процессов, диспетчеризацию прерываний, перемещение страниц между основной памятью и диском и т.п. Модули этого слоя не принимают решений о распределении ресурсов, а только обрабатывают решения, принятые модулями вышележащих уровней. Поэтому их часто называют исполнительными механизмами для модулей верхних слоев ОС.

Менеджеры ресурсов. Модули этого слоя выполняют стратегические задачи по управлению ресурсами вычислительной системы. Это менеджеры (диспетчеры) процессов, ввода-вывода, оперативной памяти и файловой системы. Каждый менеджер ведет учет свободных и используемых ресурсов и планирует их распределение в соответствии запросами приложений.

Интерфейс системных вызовов. Это верхний слой ядра ОС, взаимодействующий с приложениями и системными утилитами, он образует прикладной программный интерфейс ОС. Функции API обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной компактной форме, без указания деталей их физического расположения.

Системное ПО у Алексеева Е.Г. подразделяется на базовое и сервисное. Системные программы предназначены для управления работой вычислительной системы, выполняют различные вспомогательные функции, такие как копирования, форматирования, тестирования, выдачи справок и т. д.

|  |  |
| --- | --- |
| Базовое программное обеспечение включает в себя | Сервисное ПО включает в себя программы (утилиты) |
| операционные системы | диагностики |
| оболочки | антивирусные |
| сетевые операционные системы | обслуживания носителей |
|  | архивирования |
|  | обслуживания сети |

Системное программное обеспечение разрабатывается для эффективного управления прикладных программ. Особое место среди десятка тысяч системных программ занимают операционные системы, обеспечивающие управление всеми ресурсами компьютера для их эффективного использования.

Рассмотрим более подробно программы, так называемые утилиты.

Утили́та (utility или tool) – компьютерная программа, расширяющая стандартные возможности оборудования и операционных систем, выполняющая узкий круг специфических задач.

Эти программы дают возможность получить доступ к таким возможностям как: параметры, установки, настройки, которые недоступны без применения утилит, либо они делают этот процессы более простым и понятным, автоматизируя его. Часто утилиты входят состав операционной системы или же входят в комплект, с каким либо оборудованием, допустим клавиатурой.

Функции утилит:

1) Мониторинг показателей датчиков и производительности оборудования; мониторинг температур процессора, видеоадаптера; чтение S.M.A.R.T. жёстких дисков; бенчмарки.

2) Управление параметрами оборудования - ограничение максимальной скорости вращения CD-привода; изменение скорости вращения кулеров.

3) Контроль показателей - проверка ссылочной целостности; правильности записи данных.

4) Расширение возможностей - форматирование и/или переразметка диска с сохранением данных, удаление без возможности восстановления[4] .

Существует огромное множество типов утилит. Среди которых:

- программы, использующиеся для контроля, тестирования и диагностики правильности работы устройств компьютера, а также для обнаружения неисправностей в процессе его эксплуатации;

- программы-драйверы, расширяющие возможности операционной системы, с помощью которых возможно подключить новые устройства;

- программы-упаковщики, они же архиваторы, позволяющие записывать информацию на дисках более плотно;

- антивирусные программы, которые предназначены для предотвращения заражения компьютеров вирусами и ликвидации последствий их заражения;

- программы для оптимизации и контроля качества дискового пространства;

- программы восстановления информации, форматирования и защиты данных;

- коммуникационные программы, организующие обмен информацией между компьютерами;

- программы для управления памятью, обеспечивающие более гибкое использование оперативной памяти;

- программы для записи CD-ROM, CD-R и многие другие.

Рассмотрев данный параграф, мы увидели, что системное ПО делится на базовое и системное, которые в свою очередь тоже делятся на разделы, образуя структуру. Узнали важную роль утилит в программном обеспечение и их функции.

API (Application Programming Interface) - интерфейс прикладного программирования, .

Интерфейс между операционной системой и программами определяется набором системных вызовов.

Например, если пользовательскому процессу необходимо считать данные из файла, он должен выполнить команду системного вызова, т.е. выполнить прерывание с переключением в режим ядра и активизировать функцию операционной системы для считывания данных из файла.

Рассмотрим наиболее часто применяемых системных вызовов стандарта POSIX. В POSIX существует более 100 системных вызовов.

|  |  |  |
| --- | --- | --- |
| fork - создание нового процесса | read - читает данные из файла в буфер | link - создает ссылку |
| exit - завершение процесса | write - пишет данные из буфера в файл | unlink - удаляет ссылку |
| open - открывает файл | mkdir - создает новый каталог | mount - монтирует файловую систему |
| close - закрывает файл | rmdir - удаляет каталог | umount - демонтирует файловую систему |
|  |  | chdir - изменяет рабочий каталог |

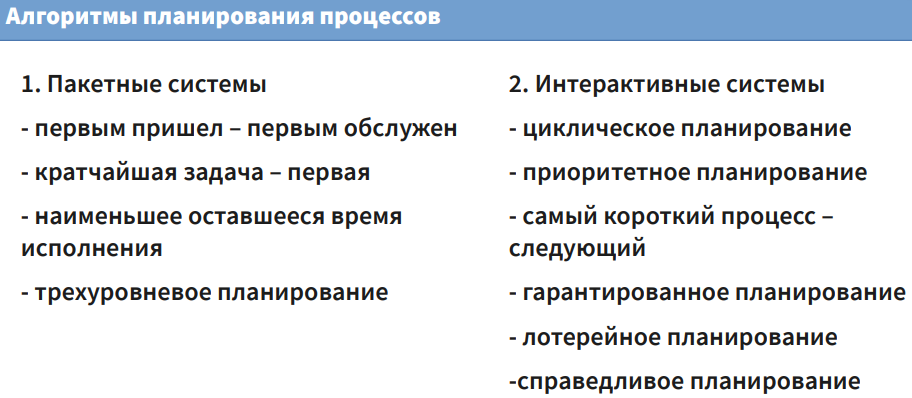
В UNIX вызовы почти один к одному идентичны библиотечным процедурам, которые используются для обращения к системным вызовам.

Рассмотрим интерфейс прикладного программирования для Windows - Win32 API. Win32 API отделен от системных вызовов. Это позволяет в разных версиях менять системные вызовы, не переписывая программы.

Поэтому непонятно является ли вызов системным (выполняется ядром), или он обрабатывается в пространстве пользователя.

В Win32 API существует более 1000 вызовов. Такое количество связано и с тем, что графический интерфейс пользователя UNIX запускается в пользовательском режиме, а в Windows встроен в ядро. Поэтому Win32 API имеет много вызовов для управления окнами, текстом, шрифтами т.д.

1. **Изобразить систему очередей планирования и объяснить принцип работы.**



4.1 Основные понятия планирования процессов

Планирование - обеспечение поочередного доступа процессов к одному процессору.

Планировщик - отвечающая за это часть операционной системы.

Алгоритм планирования - используемый алгоритм для планирования.

Ситуации, когда необходимо планирование:

* Когда создается процесс
* Когда процесс завершает работу
* Когда процесс блокируется на операции ввода/вывода, семафоре, и т.д.
* При прерывании ввода/вывода.

Алгоритм планирования без переключений (неприоритетный) - не требует прерывание по аппаратному таймеру, процесс останавливается только когда блокируется или завершает работу.

Алгоритм планирования с переключениями (приоритетный) - требует прерывание по аппаратному таймеру, процесс работает только отведенный период времени, после этого он приостанавливается по таймеру, чтобы передать управление планировщику.

Необходимость алгоритма планирования зависит от задач, для которых будет использоваться операционная система.

Основные три системы:

* Системы пакетной обработки - могут использовать неприоритетный и приоритетный алгоритм (например: для расчетных программ).
* Интерактивные системы - могут использовать только приоритетный алгоритм, нельзя допустить чтобы один процесс занял надолго процессор (например: сервер общего доступа или персональный компьютер).
* Системы реального времени - могут использовать неприоритетный и приоритетный алгоритм (например: система управления автомобилем).

Задачи алгоритмов планирования:

Для всех систем

* Справедливость - каждому процессу справедливую долю процессорного времени
* Контроль над выполнением принятой политики
* Баланс - поддержка занятости всех частей системы (например: чтобы были заняты процессор и устройства ввода/вывода)

Системы пакетной обработки

* Пропускная способность - количество задач в час
* Оборотное время - минимизация времени на ожидание обслуживания и обработку задач.
* Использование процесса - чтобы процессор всегда был занят.

Интерактивные системы

* Время отклика - быстрая реакция на запросы
* Соразмерность - выполнение ожиданий пользователя (например, пользователь не готов к долгой загрузке системы)

Системы реального времени

* Окончание работы к сроку - предотвращение потери данных
* Предсказуемость - предотвращение деградации качества в мультимедийных системах (например: потерь качества звука должно быть меньше чем видео)

4.2 Планирование в системах пакетной обработки

4.2.1 "Первый пришел - первым обслужен" (FIFO - First In Fist Out)

Процессы ставятся в очередь по мере поступления.

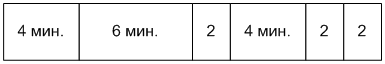
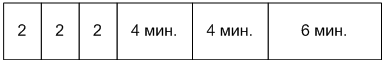
Преимущества:

* Простота
* Справедливость (как в очереди покупателей, кто последний пришел, тот оказался в конце очереди)

Недостатки:

* Процесс, ограниченный возможностями процессора может затормозить более быстрые процессы, ограниченные устройствами ввода/вывода.

4.2.2 "Кратчайшая задача - первая"

Нижняя очередь выстроена с учетом этого алгоритма

Преимущества:

* Уменьшение оборотного времени
* Справедливость (как в очереди покупателей, кто без сдачи проходит в перед)

Недостатки:

* Длинный процесс занявший процессор, не пустит более новые краткие процессы, которые пришли позже.

4.2.3 Наименьшее оставшееся время выполнение

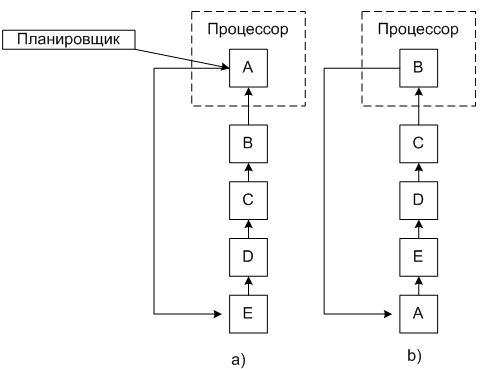
Аналог предыдущего, но если приходит новый процесс, его полное время выполнения сравнивается с оставшимся временем выполнения текущего процесса.

4.3 Планирование в интерактивных системах

4.3.1 Циклическое планирование

Самый простой алгоритм планирования и часто используемый.

Каждому процессу предоставляется квант времени процессора. Когда квант заканчивается процесс переводится планировщиком в конец очереди. При блокировке процессор выпадает из очереди.



Пример циклического планирования

Преимущества:

* Простота
* Справедливость (как в очереди покупателей, каждому только по килограмму)

Недостатки:

* Если частые переключения (квант - 4мс, а время переключения равно 1мс), то происходит уменьшение производительности.
* Если редкие переключения (квант - 100мс, а время переключения равно 1мс), то происходит увеличение времени ответа на запрос.

4.3.2 Приоритетное планирование

Каждому процессу присваивается приоритет, и управление передается процессу с самым высоким приоритетом.

Приоритет может быть динамический и статический.

Динамический приоритет может устанавливаться так:

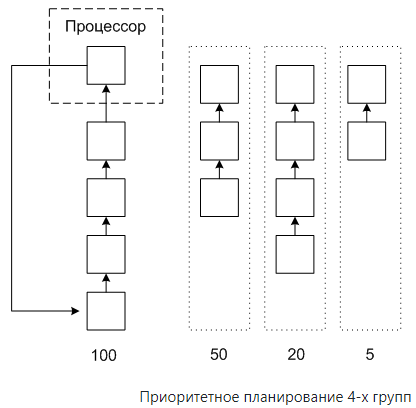
П=1/Т, где Т- часть использованного в последний раз кванта

Если использовано 1/50 кванта, то приоритет 50.

Если использован весь квант, то приоритет 1.

Т.е. процессы, ограниченные вводом/вывода, будут иметь приоритет над процессами, ограниченными процессором.

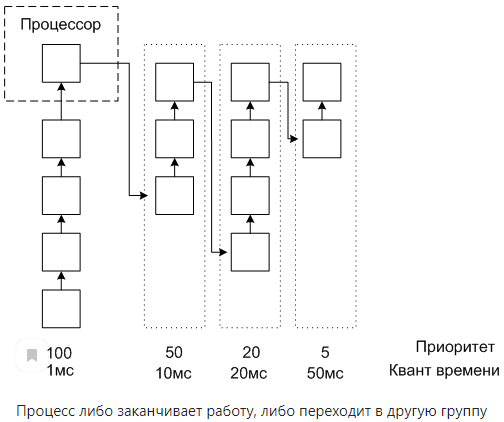
Часто процессы объединяют по приоритетам в группы, и используют приоритетное планирование среди групп, но внутри группы используют циклическое планирование.



4.3.3 Методы разделения процессов на группы

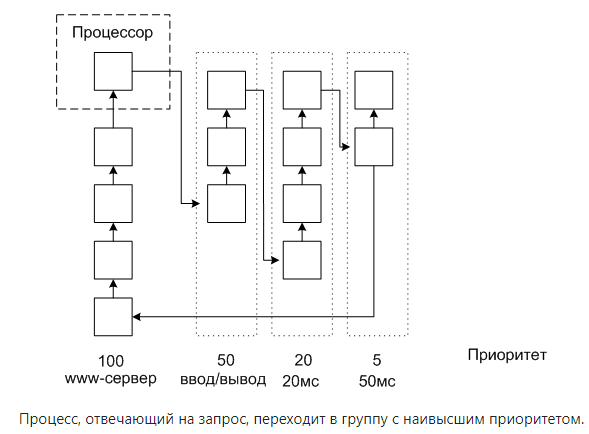
Группы с разным квантом времени

Сначала процесс попадает в группу с наибольшим приоритетом и наименьшим квантом времени, если он использует весь квант, то попадает во вторую группу и т.д. Самые длинные процессы оказываются в группе наименьшего приоритета и наибольшего кванта времени.



Этот метод напоминает алгоритм - "Кратчайшая задача - первая".

Группы с разным назначением процессов



Такой механизм позволяет повысить приоритет работы с клиентом.

Гарантированное планирование

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

Лотерейное планирование

Процессам раздаются "лотерейные билеты" на доступ к ресурсам. Планировщик может выбрать любой билет, случайным образом. Чем больше билетов у процесса, тем больше у него шансов захватить ресурс.

Справедливое планирование

Процессорное время распределяется среди пользователей, а не процессов. Это справедливо если у одного пользователя несколько процессов, а у другого один.

4.4 Планирование в системах реального времени

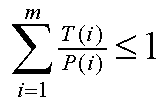
Системы реального времени делятся на:

* жесткие (жесткие сроки для каждой задачи) - управление движением
* гибкие (нарушение временного графика не желательны, но допустимы) - управление видео и аудио

Внешние события, на которые система должна реагировать, делятся:

* периодические - потоковое видео и аудио
* непериодические (непредсказуемые) - сигнал о пожаре

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.

4.4.1 Планирование однородных процессов

В качестве однородных процессов можно рассмотреть видео сервер с несколькими видео потоками (несколько пользователей смотрят фильм).

Т.к. все процессы важны, можно использовать циклическое планирование.

Но так как количество пользователей и размеры кадров могут меняться, для реальных систем он не подходит.

4.4.2 Общее планирование реального времени

Используется модель, когда каждый процесс борется за процессор со своим заданием и графиком его выполнения.

Планировщик должен знать:

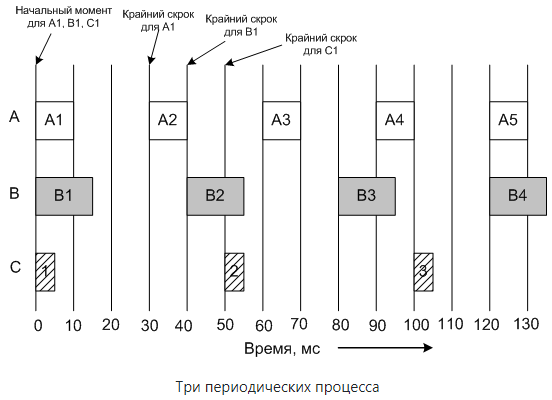
* частоту, с которой должен работать каждый процесс
* объем работ, который ему предстоит выполнить
* ближайший срок выполнения очередной порции задания

Рассмотрим пример из трех процессов.

Процесс А запускается каждые 30мс, обработка кадра 10мс

Процесс В частота 25 кадров, т.е. каждые 40мс, обработка кадра 15мс

Процесс С частота 20 кадров, т.е. каждые 50мс, обработка кадра 5мс



Проверяем, можно ли планировать эти процессы.

10/30+15/40+5/50=0.808<1

Условие выполняется, планировать можно.

Будем планировать эти процессы статическим (приоритет заранее назначается каждому процессу) и динамическим методами.

4.4.3 Статический алгоритм планирования RMS (Rate Monotonic Scheduling)

Процессы должны удовлетворять условиям:

* Процесс должен быть завершен за время его периода
* Один процесс не должен зависеть от другого
* Каждому процессу требуется одинаковое процессорное время на каждом интервале
* У непериодических процессов нет жестких сроков
* Прерывание процесса происходит мгновенно

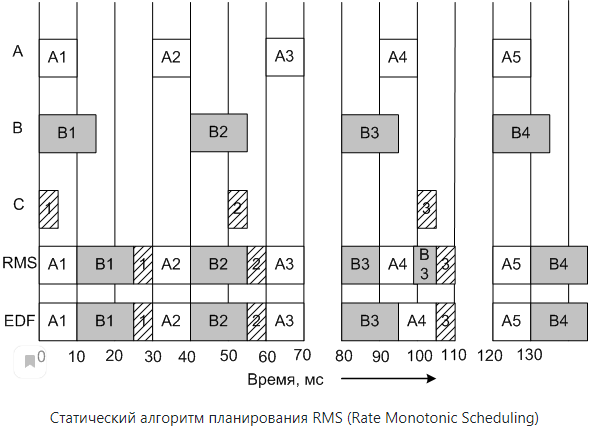
Приоритет в этом алгоритме пропорционален частоте.

Процессу А он равен 33 (частота кадров)

Процессу В он равен 25

Процессу С он равен 20

Процессы выполняются по приоритету.



4.4.4 Динамический алгоритм планирования EDF (Earliest Deadline First)

Наибольший приоритет выставляется процессу, у которого осталось наименьшее время выполнения.

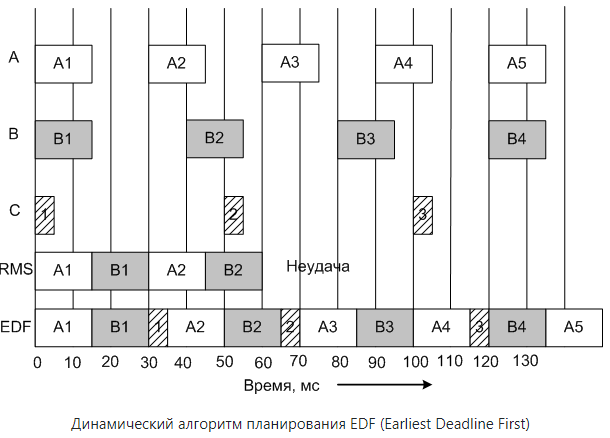
При больших загрузках системы EDF имеет преимущества.

Рассмотрим пример, когда процессу А требуется для обработки кадра - 15мс.

Проверяем, можно ли планировать эти процессы.

15/30+15/40+5/50=0.975<1

Загрузка системы 97.5%



Алгоритм планирования RMS терпит неудачу.

1. **Приведите основные стратегии планировщиков.**

(см. предыдущий вопрос)

1. **Описать процесс свопинга. Перечислить достоинства и недостатки.**

6.3 Методы с использованием внешней памяти (свопинг и виртуальная память)

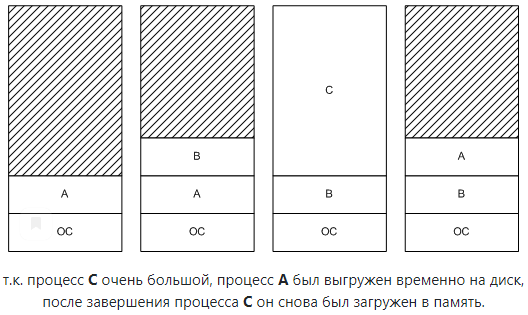
Так как памяти, как правило, не хватает. Для выполнения процессов часто приходится использовать диск.

Основные способы использования диска:

* Свопинг (подкачка) - процесс целиком загружается в память для работы
* Виртуальная память - процесс может быть частично загружен в память для работы

6.3.1 Свопинг (подкачка)

При нехватке памяти процессы могут быть выгружены на диск.



Как мы видим процесс А второй раз загрузился в другое адресное пространство, должны создаваться такие условия, которые не повлияют на работу процесса.

Свопер - планировщик, управляющий перемещением данных между памятью и диском.

Этот метод был основным для UNIX до версии 3BSD.

Управление памятью с помощью битовых массивов

Вся память разбивается на блоки (например, по 32бита), массив содержит 1 или 0 (занят или незанят).

Чтобы процессу в 32Кбита занять память, нужно набрать последовательность из 1000 свободных блоков.

Такой алгоритм займет много времени.

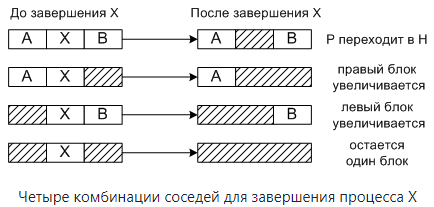


Управление памятью с помощью связных списков

Этот способ отслеживает списки занятых (между процессами) и свободных (процессы) фрагментов памяти.

Запись в списке указывает на:

* занят (P) или незанят (H) фрагмент
* адрес начала фрагмента
* длину фрагмента



Алгоритмы выделения блока памяти:

* первый подходящий участок.
* следующий подходящий участок, стартует не сначала списка, а с того места на котором остановился в последний раз.
* самый подходящий участок (медленнее, но лучше использует память).
* самый неподходящий участок, расчет делается на то, что программа займет самый большой участок, а лишнее будет отделено в новый участок, и он будет достаточно большой для другой программы.

https://moodle.kstu.ru/mod/page/view.php?id=57

1. **Сформулируйте основные функции операционных систем.**

Основные функции (простейшие ОС):

* Загрузка приложений в оперативную память и их выполнение;
* Стандартизованный доступ к периферийным устройствам (устройства ввода-вывода);
* Управление оперативной памятью (распределение между процессами, виртуальная память);
* Управление доступом к данным на энергонезависимых носителях (таких как Жёсткий диск, Компакт-диск и т. д.), как правило с помощью файловой системы;
* Пользовательский интерфейс;
* Сетевые операции, поддержка стека протоколов

Дополнительные функции:

* Параллельное или псевдопараллельное выполнение задач (многозадачность);
* Взаимодействие между процессами: обмен данными, взаимная синхронизация;
* Защита самой системы, а также пользовательских данных и программ от злонамеренных действий пользователей или приложений;
* Разграничение прав доступа и многопользовательский режим работы (аутентификация, авторизация).

1. **Приведите классификацию ресурсов вычислительной системы.**

С точки зрения вычислительных систем, ресурсом является средство вычислительной системы, которое может быть выделено процессу (задаче) на определённый период времени для его успешного развития и выполнения.

Для более отчётливого понимания сущности ресурсов необходимо их определять в соответствии с различными классификационными признаками. Рассмотрим некоторые из них.

Реальность существования.

В соответствии с этим классификационным признаком все ресурсы вычислительной системы можно разделить на физические и виртуальные.

Физический ресурс – это ресурс, который реально существует и имеет конкретные характеристики. Виртуальный (мнимый) ресурс представляет собой некоторую модель, в основе которой лежит соответствующий физический ресурс. Как модель, виртуальный ресурс, реализуется в некоторой программно-аппаратной форме, и в этом смысле он существует. Виртуальный ресурс может иметь свойства и характеристики, значительно отличающиеся от свойств и характеристик соответствующего физического ресурса. Примером такого отличия является разница в характеристиках и в возможностях виртуальной и физической памяти компьютера.

Форма реализации.

В соответствии с этим признаком ресурсы разделяются на аппаратные (твёрдые) и мягкие (остальные). К мягким ресурсам относятся, прежде всего, программные ресурсы. К аппаратным ресурсам относятся процессор, оперативная память, магнитные диски и другие устройства ввода/вывода, а к программным – библиотечные модули, драйверы, компиляторы и т. д.

Время существования

Ресурс, существующий в системе до момента порождения процесса и доступный для использования на всём интервале существования процесса, называется постоянным. Временной ресурс может появляться и исчезать в системе динамически в течении интервала существования процесса. К временным ресурсам относятся различного рода сигналы, включая сигналы прерываний, а также различные сообщения, которыми процессы обмениваются между собой.

Структура.

Ресурс называется простым, если не содержит составных элементов и рассматривается при распределении процессам как единое целое. Составной ресурс характеризуется некоторой структурой, т. е. имеет в своём составе некоторые однотипные элементы, обладающие одинаковыми характеристиками. Простые ресурсы могут находиться только в двух состояниях – “занят”, “свободен”, а для составных ресурсов характерны три состояния – “занят” , “свободен”, “частично занят”. Примером простого ресурса, с точки зрения операционных систем, является процессор, а составного – оперативная память.

Возможность восстанавливаемости.

+Ресурс называется воспроизводимым (системным), если после его использования некоторым процессом, он может быть использован другим процессом. Потребляемый ресурс – это ресурс, который после своего использования некоторым процессом исчезает из системы. К потребляемым ресурсам, прежде всего, относятся сигналы и сообщения.

1. **Опишите эволюцию структур операционных систем. Приведите структурные схемы.**

История ОС насчитывает примерно полвека. Она во многом определялась и определяется развитием элементной базы и вычислительной аппаратурой.

Первое поколение.

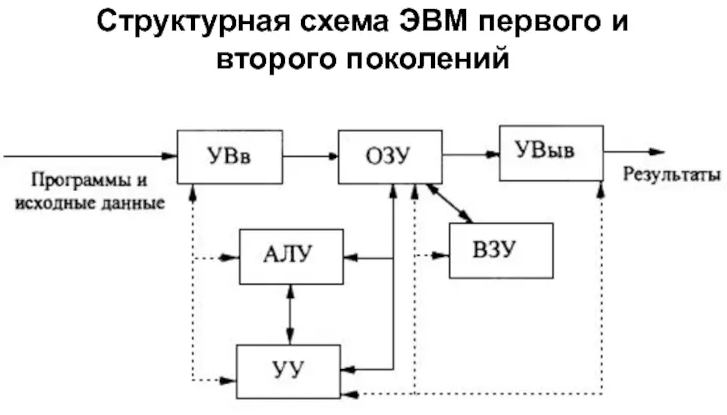
40-е годы. Первые цифровые вычислительные машины без ОС. Организация вычислительного процесса решается программистом с пульта управления.

Второе поколение.

50-е годы. Появление прообраза ОС - мониторные системы, реализующие систему пакетной обработки заданий.

Пакетный режим

Необходимость оптимального использования дорогостоящих вычислительных ресурсов привела к появлению концепции «пакетного режима» исполнения программ. Пакетный режим предполагает наличие очереди программ на исполнение, причём ОС может обеспечивать загрузку программы с внешних носителей данных в оперативную память, не дожидаясь завершения исполнения предыдущей программы, что позволяет избежать простоя процессора.



Третье поколение.

1965-1980 г.г. Переход к интегральным схемам. IBM/360. Реализованы практически все основные концепции, присущие современным ОС: разделение времени и многозадачность, разделение полномочий, реальный масштаб времени, файловые структуры и файловые системы. Реализация мультипрограммирования потребовала внесения очень важных изменений в аппаратуру компьютера: привилегированный и пользовательский режимы, средства защиты областей памяти, развитой системы прерываний.

Разделение времени и многозадачность

Уже пакетный режим в своём развитом варианте требует разделения процессорного времени между выполнением нескольких программ. Необходимость в разделении времени (многозадачности, мультипрограммировании) проявилась ещё сильнее при распространении в качестве устройств ввода-вывода телетайпов (а позднее, терминалов с электронно-лучевыми дисплеями) (1960-е годы). Поскольку скорость клавиатурного ввода (и даже чтения с экрана) данных оператором много ниже, чем скорость обработки этих данных компьютером, использование компьютера в «монопольном» режиме (с одним оператором) могло привести к простою дорогостоящих вычислительных ресурсов.

Разделение времени позволило создать «многопользовательские» системы, в которых один (как правило) центральный процессор и блок оперативной памяти соединялся с многочисленными терминалами. При этом часть задач (таких, как ввод или редактирование данных оператором) могла исполняться в режиме диалога, а другие задачи (такие, как массивные вычисления) — в пакетном режиме.

Разделение полномочий

Распространение многопользовательских систем потребовало решения задачи разделения полномочий, позволяющей избежать возможности модификации исполняемой программы или данных одной программы в памяти компьютера другой (содержащей ошибку или злонамеренно подготовленной) программы, а также модификации самой ОС прикладной программой.

Реализация разделения полномочий в ОС была поддержана разработчиками процессоров, предложивших архитектуры с двумя режимами работы процессора — «реальным» (в котором исполняемой программе доступно всё адресное пространство компьютера) и «защищённым» (в котором доступность адресного пространства ограничена диапазоном, выделенном при запуске программы на исполнение).

Реальный масштаб времени

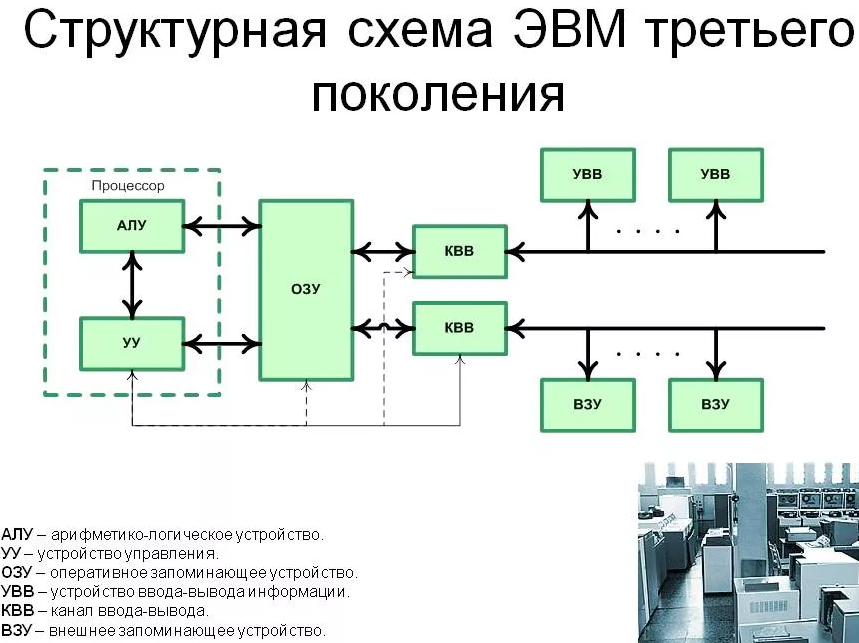
Применение универсальных компьютеров для управления производственными процессами потребовало реализации «реального масштаба времени» («реального времени») — синхронизации исполнения программ с внешними физическими процессами.

Включение функции реального масштаба времени в ОС позволило создавать системы, одновременно обслуживающие производственные процессы и решающие другие задачи (в пакетном режиме и (или) в режиме разделения времени).

Такие операционные системы получили название Операционные системы с планированием в реальном масштабе времени или сокращенно ОСРВ.

Файловые системы и структуры

Постепенная замена носителей с последовательным доступом (перфолент, перфокарт и магнитных лент) накопителями произвольного доступа (на магнитных диск)



Четвертое поколение.

Конец 70-х. Создан рабочий вариант стека протоколов TCP/IP. В 1983 году он был стандартизирован. Независимость от производителей, гибкость и эффективность, доказанные успешной работой Интернет, сделала этот стек протоколов основным стеком для большинства ОС.

Начало 80-х. Появление персональных компьютеров. Бурный рост локальных сетей. Поддержка сетевых функций стала необходимым условием. 80-е годы. Приняты основные стандарты на коммуникационные технологии локальных сетей: Ethernet, Token Ring, FDDI. Это позволило обеспечить совместимость сетевых ОС на нижних уровнях.

Начало 90-х. Практически все ОС стали сетевыми. Появились специализированные сетевые ОС (например IOS, работающая в маршрутизаторах)

Последнее десятилетие. Особое внимание корпоративным сетевым ОС, для которых характерны высокая степень масштабируемости, поддержка сетевой работы, развитые средства обеспечения безопасности, способность работать в гетерогенной среде, наличие средств централизованного администрирования.



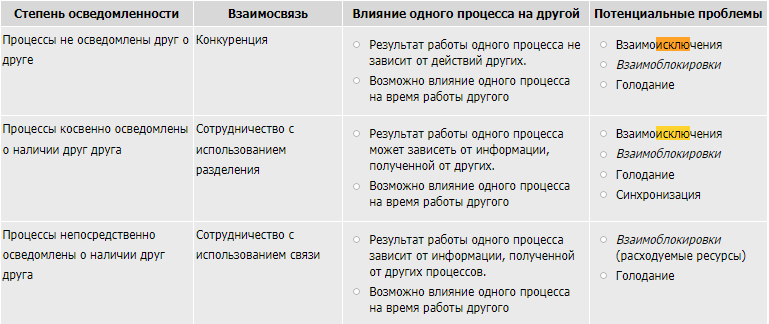
1. **Опишите процесс синхронизация и взаимодействие процессов. Эффект «гонок». Критическая секция. Взаимное исключение. Способы обеспечения взаимного исключения.**

В мультипрограммных однопроцессорных системах процессы чередуются, обеспечивая эффективное выполнение программ. В многопроцессорных системах возможно не только чередование, но и перекрытие процессов. Обе эти технологии, которые можно рассматривать как примеры параллельных вычислений, порождают одинаковые проблемы. Выполнение процессов и потоков в мультипрограммной среде всегда имеет асинхронный характер – невозможно предсказать относительную скорость выполнения процессов. Момент прерывания потоков, время нахождения их в очередях к разделяемым ресурсам, порядок выбора потоков для выполнения – все эти события являются результатом стечения многих обстоятельств и являются случайными, это справедливо как по отношению к потокам одного процесса, выполняющим общий программный код, так и по отношению к потокам разных процессов, каждый из которых выполняет собственную программу.

Способы взаимодействия процессов (потоков) можно классифицировать по степени осведомленности одного процесса о существовании другого [10].

* Процессы не осведомлены о наличии друг друга (например, процессы разных заданий одного или различных пользователей). Это независимые процессы, не предназначенные для совместной работы. Хотя эти процессы и не работают совместно, ОС должна решать вопросы конкурентного использования ресурсов. Например, два независимых приложения могут затребовать доступ к одному и тому же диску или принтеру. ОС должна регулировать такие обращения.
* Процессы косвенно осведомлены о наличии друг друга (например, процессы одного задания). Эти процессы не обязательно должны быть осведомлены о наличии друг друга с точностью до идентификатора процесса, однако они разделяют доступ к некоторому объекту, например, буферу ввода-вывода, файлу или БД. Такие процессы демонстрируют сотрудничество при разделении общего объекта.
* Процессы непосредственно осведомлены о наличии друг друга (например, процессы, работающие последовательно или поочередно в рамках одного задания). Такие процессы способны общаться один с другим с использованием идентификаторов процессов и изначально созданы для совместной работы. Эти процессы также демонстрируют сотрудничество при работе.

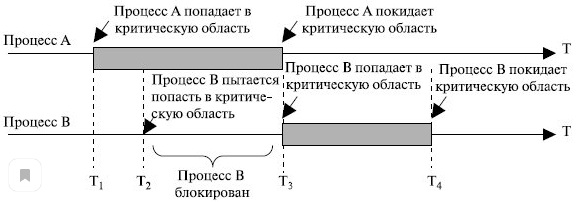
Таким образом, потенциальные проблемы, связанные с взаимодействием и синхронизацией процессов и потоков, могут быть представлены следующей таблицей.



При необходимости использовать один и тот же ресурс параллельные процессы вступают в конфликт (конкурируют) друг с другом. Каждый из процессов не подозревает о наличии остальных и не повергается никакому воздействию с их стороны. Отсюда следует, что каждый процесс не должен изменять состояние любого ресурса, с которым он работает. Примерами таких ресурсов могут быть устройства ввода-вывода, память, процессорное время, часы.

Между конкурирующими процессами не происходит никакого обмена информацией. Однако выполнение одного процесса может повлиять на поведение конкурирующего процесса. Это может, например, выразиться в замедлении работы одного процесса, если ОС выделит ресурс другому процессу, поскольку первый процесс будет ждать завершения работы с этим ресурсом. В предельном случае блокированный процесс может никогда не получить доступ к нужному ресурсу и, следовательно, никогда не сможет завершиться.

В случае конкурирующих процессов (потоков) возможно возникновение трех проблем. Первая из них – необходимость взаимных исключений (mutual exclusion). Предположим, что два или большее количество процессов требуют доступ к одному неразделяемому ресурсу, как например принтер (рис. 5.12). О таком ресурсе будем говорить как о критическом ресурсе, а о части программы, которая его использует, – как о критическом разделе (critical section) программы. Крайне важно, чтобы в критической ситуации в любой момент могла находиться только одна программа. Например, во время печати файла требуется, чтобы отдельный процесс имел полный контроль над принтером, иначе на бумаге можно получить чередование строк двух файлов.



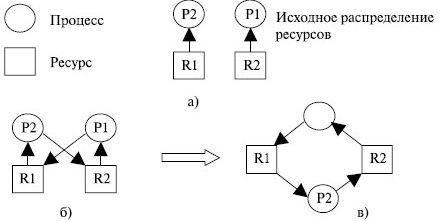
Критическая секция

Осуществление взаимных исключений создает две дополнительные проблемы. Одна из них – взаимоблокировки (deadlock) или тупики. Рассмотрим, например, два процесса – P1 и P2, и два ресурса – R1 и R2. Предположим, что каждому процессу для выполнения части своих функций требуется доступ к общим ресурсам. Тогда возможно возникновение следующей ситуации: ОС выделяет ресурс R1 процессу Р2, а ресурс R2 – процессу Р1. В результате каждый процесс ожидает получения одного из двух ресурсов. При этом ни один из них не освобождает уже имеющийся ресурс, ожидая получения второго ресурса для выполнения функций, требующих наличие двух ресурсов. В результате процессы оказываются взаимно заблокированными.

Очень удобно моделировать условия возникновения тупиков, используя направленные графы [17] (предложено Holt, 1972). Графы имеют 2 вида узлов: процессы-кружочки и ресурсы-квадратики. Ребро, направленное от квадрата (ресурса) к кружку (процессу), означает, что ресурс был запрошен, получен и используется. В нашем примере это будет изображено так, как показано на рис. 5.13 а).

Ребро, направленное от процесса (кружка) к ресурсу (квадрату), означает, что процесс в данный момент заблокирован и находится в состоянии ожидания доступа к этому ресурсу. В нашем примере граф надо достроить, как показано на рис. 5.13 б) или в). Цикл в графе означает наличие взаимной блокировки процессов.

Существует еще одна проблема у конкурирующих процессов – голодание. Предположим, что имеется 3 процесса ( Р1, Р2, Р3 ), каждому из которых периодически требуется доступ к ресурсам R. Представим ситуацию, в которой Р1 обладает ресурсом, а Р2 и Р3 приостановлены в ожидании освобождения ресурса R. После выхода Р1 из критического раздела доступ к ресурсу будет получен одним из процессов Р2 или Р3.



Тупиковая ситуация

Пусть ОС предоставила доступ к ресурсу процессу Р3. Пока он работает с ресурсом, доступ к ресурсу вновь требуется процессу Р1. В результате по освобождении ресурса R процессом Р3 может оказаться, что ОС вновь предоставит доступ к ресурсу процессу Р1. Тем временем процессу Р3 вновь требуется доступ к ресурсу R. Таким образом, теоретически возможна ситуация, в которой процесс Р2 никогда не получит доступ к требуемому ему ресурсу, несмотря на то, что никакой взаимной блокировки в данном случае нет.

Рассмотрим случай сотрудничества с использованием разделения. Этот случай охватывает процессы (потоки), взаимодействующие с другими процессами (потоками), без наличия явной информации о них. Например, несколько потоков могут обращаться к разделяемым переменным (глобальным) или совместно используемым файлам или базам данных. Поскольку данные хранятся в ресурсах (устройствах памяти), в этом случае также возможны проблемы взаимоблокировок, взаимоисключения и голодания. Единственное отличие в том, что доступ к данным может осуществляться в двух режимах – чтения и записи, и взаимоисключающими должны быть только операции записи.

Однако в этом случае вносится новое требование синхронизации процессов для обеспечения согласованности данных.

Пусть имеются два процесса, представленные последовательностью неделимых (атомарных) операций:

P: a; b; c; и Q: d; e; f, где a, b, c, d, e, f – атомарные операции.

При последовательном выполнении активностей мы получаем следующую последовательность атомарных действий:

PQ: a b c d e f

Что произойдет при исполнении этих процессов псевдопараллельно, в режиме разделения времени? Процессы могут расслоиться на неделимые операции с различным их чередованием, то есть может произойти то, что на английском языке принято называть словом interleaving. Возможные варианты чередования:

а b c d e f

a b d c e f

a b d e c f

a b d e f c

a d b c e f

d e f a b c

В данном случае атомарные операции активностей могут чередоваться всевозможными способами с сохранением своего порядка расположения внутри процессов. Так как псевдопараллельное выполнение двух процессов приводит к чередованию их неделимых операций, результат псевдопараллельного выполнения может отличаться от результата последовательного выполнения. Пусть есть два процесса P и Q, состоящие из двух атомарных операций:

P: x=2; y=x-1; Q: x=3; y=x+1

Что мы получим в результате их псевдопараллельного выполнения, если переменные x и y являются общими для процессов? Легко видеть, что возможны четыре разных набора значений для пары (x, y): (3, 4), (2, 1), (2, 3) и (3, 2). Будем говорить, что набор процессов детерминирован, если всякий раз при псевдопараллельном исполнении для одного и того же набора входных данных он дает одинаковые выходные данные. В противном случае он недетерминирован. Выше приведен пример недетерминированного набора программ. Понятно, что детерминированный набор активностей можно безбоязненно выполнять в режиме разделения времени. Для недетерминированного набора такое исполнение нежелательно.

Можно ли до получения результатов, заранее, определить, является ли набор активностей детерминированным или нет? Для этого существуют достаточные условия Бернстайна [17]. Изложим их применительно к программам с разделяемыми переменными.

Введем наборы входных и выходных переменных программы. Для каждой атомарной операции наборы входных и выходных переменных – это наборы переменных, которые атомарная операция считывает и записывает. Набор входных переменных программы R(P) ( R от слова read ) суть объединение наборов входных переменных для всех ее неделимых действий. Аналогично, набор выходных переменных программы W(P) ( W от слова write ) суть объединение наборов выходных переменных для всех ее неделимых действий. Например, для программы

P: x = u + v; y = x \* w;

получаем R(P) = {u, v, x, w}, W(P) = {x, y}. Заметим, что переменная x присутствует как в R(P), так и в W(P).

Теперь сформулируем условия Бернстайна.

Если для двух данных процессов P и Q:

пересечение W(P) и W(Q) пусто,

пересечение W(P) с R(Q) пусто,

пересечение R(P) и W(Q) пусто,

тогда выполнение P и Q детерминировано.

Если эти условия не соблюдены, возможно, что параллельное выполнение P и Q детерминировано, но возможно, что и нет. Случай двух процессов естественным образом обобщается на их большее количество.

Условия Бернстайна информативны, но слишком жестки. По сути дела, они требуют практически невзаимодействующих процессов. Однако хотелось бы, чтобы детерминированный набор образовывал процессы, совместно использующие информацию и обменивающиеся ею. Для этого нам необходимо ограничить число возможных чередований атомарных операций, исключив некоторые чередования с помощью механизмов синхронизации выполнения программ и обеспечив тем самым упорядоченный доступ программ к некоторым данным.

Про недетерминированный набор программ говорят, что он имеет race condition (состояние гонки, состояние состязания). В приведенном выше примере процессы состязаются за вычисление значений переменных x и y.

Задачу упорядоченного доступа к разделяемым данным (устранение race condition), в том случае, если нам не важна его очередность, можно решить, если обеспечить каждому процессу эксклюзивное право доступа к этим данным. Каждый процесс, обращающийся к разделяемым ресурсам, исключает для всех других процессов возможность одновременного с ним общения с этими ресурсами, если это может привести к недетерминированному поведению набора процессов. Такой прием называется взаимоисключением (mutual exclusion). Если очередность доступа к разделяемым ресурсам важна для получения правильных результатов, то одними взаимоисключениями уже не обойтись.

При сотрудничестве с использованием связи различные процессы принимают участие в общей работе, которая их объединяет. Связь обеспечивает возможность синхронизации, или координации, различных действий процессов. Обычно можно считать, что связь состоит из сообщений определенного вида. Примитивы для отправки и получения сообщений могут быть предоставлены языком программирования или ядром операционной системы.

Поскольку в процессе передачи сообщений не происходит какого-либо совместного использования ресурсов, взаимоисключения не требуется, хотя проблемы взаимоблокировок и голодания остаются актуальными. В качестве примера взаимоблокировки можно привести ситуацию, при которой каждый из двух процессов заблокирован ожиданием сообщения от другого процесса. Голодание можно проиллюстрировать следующим образом. Пусть есть три процесса Р1, Р2, Р3, а те, в свою очередь, пытаются связаться с процессом Р1. Может возникнуть ситуация, когда Р1 и Р2 постоянно связываются друг с другом, а Р3 остается заблокированным, ожидая связи с процессом Р1.

1. **Изобразите схему преобразования адреса для платформы Intel. Опишите элемент PTE.**

Windows NT построена в соответствии с классическими принципами:

ВП имеет страничную организацию.

В общем виде схема описывается следующим образом:

Линейный адрес разбивается на несколько частей:

Старшая часть адреса содержит номер элемента в корневой таблице. Этот элемент содержит адрес в таблице следующего уровня. Следующая часть линейного адреса содержит номер элемента уже в этой таблице и т.д. до последней страницы, которая содержит элемент физической страницы.

Самая младшая часть страницы является номером байта в этой физической таблице страниц.

Размер страниц для платформы фирмы Intel составляет 4 Кб, а для платформ DEC Alpha 8 Кб. А схема страничного преобразования выглядит так:

Линейный 32-х разрядный адрес занимает три части:

Старшие 10 разрядов адреса определяют номер одного из 1024 элементов в каталоге страниц, адрес которого находится в регистре процессора CR3. Этот элемент содержит физический адрес таблицы страниц. Следующие 10 разрядов линейного адреса определяют номер элемента таблицы. Элемент в свою очередь содержит физический адрес страницы ВП. Младших 12 разрядов линейного адреса достаточно, чтобы определить внутри 4КБ страницы точный физический номер, адресуемой ячейки.

Рассмотрим отдельный элемент таблицы страницы, которой называют PTE (Page Table Element).

Старшие 5 бит определяют тип страницы с точки зрения допустимых операций. WIN 32 API поддерживает 3 допустимых значения этого поля:

PAGE NO Access.

PAGE READ ONLY.

PAGE READ/WRITE.

Следующие 20 бит определяют базовый физический адрес в странице памяти и если их до 12 битами младшими разрядами линейного адреса, то они образуют физический адрес ячейки памяти, к которой производится обращение. Следующие 4 бита (PTE) описывают используемый файл подкачки. Комбинация этих битов определяют один из 16 возможных в системе файлов. Последующие 3 бита определяют состояние страницы в системе.

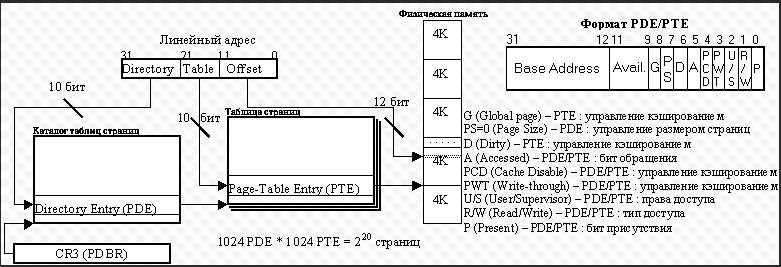
Старший из них называют T – Transition определяет следующую страницу, как переходную.

D – Dirty отмечает страницу, в которой была произведена запись. Информация об изменениях в страницах необходима для того, чтобы принимать решение о состоянии страницы в файле по указанию при её вытеснении. То есть если страница не изменялась в памяти после загрузки, то её можно просто стереть, так как копия осталась в файле подкачки.

Р – Present младший бит определяет присутствует страница в ОП или в файле подкачки. Для ускорения страничного преобразования в процессоре имеется специальная кэш-память, которая называется TLB (Translation Lookside Buffer). В ней хранятся наиболее часто используемые элементы каталога и элементы страниц. Каждый процесс в Windows NT имеет свой отдельный каталог страниц и свое собственное независимое адресное пространство, что дает дополнительные возможности при защите процессов.

Механизм преобразования виртуального адреса в физический при страничной организации памяти.

Схема преобразования виртуального адреса в физический для сегментно-страничной организации памяти.



1. **Перечислите типы операционных систем и их основные отличительные свойства.**

Все огромное множество существующих операционных систем можно разделить на четыре основных типа:

Операционные системы реального времени (Real-time operating system, RTOS). Операционные системы реального времени предназначены для управления машинным оборудованием, научными инструментами и промышленными системами. Обычно пользовательский интерфейс таких систем не балует дизайнерскими изысками, а утилиты, ориентированные на конечного пользователя, отсутствуют. Это готовая к использованию «закрытая коробка». Важнейшей задачей такого типа операционной системы является следить за тем, чтобы определенная операция выполнялась в определенный отрезок времени. И эти отрезки были равными. В сложных машинах нельзя допустить, чтобы их элементы двигались быстрее чем нужно, поскольку система располагает «лишними» ресурсами. Или, наоборот, не двигались вовсе по причине занятости системы

Однопользовательские однозадачные операционные системы (Single-user, single task). Как следует из их названия, эти операционные системы ориентированы на выполнение одним пользователем одной задачи в один период времени. Ярким примером системы такого типа может служить Palm OS для наладонников Palm. Эти устройства пользовались определенной популярностью до того как началось массовое распространение смартфонов

Однопользовательские многозадачные операционные системы (Single-user, multi-tasking). Системы этого типа управляют большинством современных настольных и портативных компьютеров. И Microsoft Windows, и Apple OS X относятся именно к этому типу. Например, пользователь Windows может одновременно писать заметку в текстовом редакторе и загружать файл из Интернета. В это же самое время принтер может печатать, а почтовый клиент принимать электронную корреспонденцию

Многопользовательские операционные системы (Multi-user). Многопользовательские операционные системы позволяют нескольким пользователям одновременно получать доступ к ресурсам устройства. Операционной системе приходится удерживать баланс между теми требованиями, которые предъявляют разные пользователи. При этом операционной системе приходится следить за тем, чтобы у каждой из работающих программ было достаточно ресурсов, и задача, решаемая одним из пользователей, не мешала другим пользователям. В качестве ярких примеров многопользовательских систем можно привести Unix, VMS и операционные системы для больших ЭВМ, подобные MVS

Важно внести ясность в некоторые тонкости, которые могут затруднить определение типа операционной системы. Существуют многопользовательские операционные системы и системы, поддерживающие сетевые соединения. И это различные операционные системы. Например, Windows 2000 и Novell Netware поддерживали сотни и даже тысячи сетевых соединений. При этом их нельзя считать истинно многопользовательскими. Единственным настоящим пользователем Windows 2000 и Netware является системный администратор. Сетевая поддержка и все удаленные аккаунты пользователей включены в общий план операционной системы, программно определяемый ее администратором.

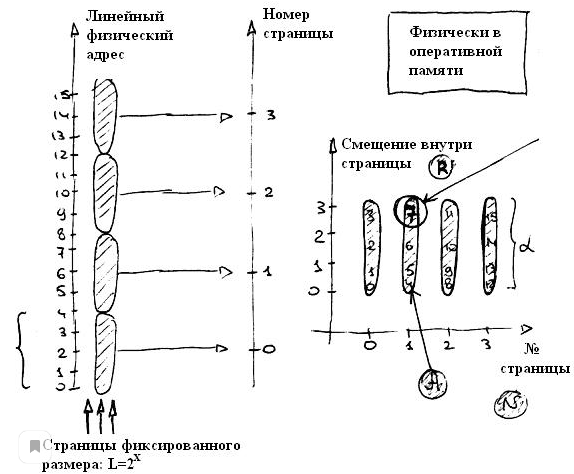
1. **Постройте схему структурирования фиксированными страницами и опишите процесс преобразования адресов.**

41. Организация виртуальной оперативной памяти. Схема структурирования фиксированными страницами.

Различают два класса схем структуризации адресного пространства:

1-Схемы страничной структуризации.

2-Схемы сегментной структуризации.



Схемы структурирования фиксированными страницами.

Если размер адресного пространства кратен 2, то и размер страницы также выбирают кратным 2. Реально в ОС 512, 1024 и выше. Это позволяет упростить механизм преобразования адресов. В каждую страницу входят одинаковое число адресов L. Для перехода из страничного в исходное одномерное адресное пространство используется выражение A=kxL+R для вычисления адреса. Например, от адреса (1,3) мы используем A=1x2^2+3=7. Чтобы перейти в непрерывный адрес нужно выполнить два действия: умножение и сложение. Можно обойтись одним действием, если воспользоваться двоичным представлением номера страницы и смещение. Из двух двоичных чисел с помощью конкатенации получают третье число. Например, (01) и (11) получается (0111)(7).

Механизм работы страницы.

Во время загрузки процесса часть его виртуальных страниц помещается в ОП, а остальные на диск. Причём, смещение виртуальной страницы не обязательно располагается в смежных физических страницах. ОС создаёт информационную структуру, которая называется таблица страниц. В этой таблице устанавливается соответствие между номерами виртуальных и физических страниц. (Но это только для страниц, загруженных в ОП). Или может делаться отметка о том, что виртуальная страница выгружена на диск. В таблице страниц содержится следующая управляющая информация:

1-Признак модификации страницы.

2-Признак запрета на выгрузку (выгрузка некоторых страниц может быть запрещена).

3-Признак обращения к странице (используются для подсчёта числа обращений за определённый период времени).

4-Время преобразования виртуального адреса в физический (в основном определяется временем доступа к страницам).

Страничная организация может быть реализована в упрощённом варианте без выгрузки страниц на диск. В этом случае фрагментация уменьшается, так как программа может загружаться в несмежные области. Но при этом расширение за границы физического пространства не происходит.

https://pandia.ru/text/78/121/98980-3.php

1. **Сформулируйте задачи и функции файловой системы.**

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы организовать эффективную работу с данными, хранящимися во внешней памяти, и обеспечить пользователю удобный интерфейс при работе с такими данными. Организовать хранение информации на магнитном диске непросто. Это требует, например, хорошего знания устройства контроллера диска, особенностей работы с его регистрами. Непосредственное взаимодействие с диском - прерогатива компонента системы ввода-вывода ОС, называемого драйвером диска. Для того чтобы избавить пользователя компьютера от сложностей взаимодействия с аппаратурой, была придумана ясная абстрактная модель файловой системы. Операции записи или чтения файла концептуально проще, чем низкоуровневые операции работы с устройствами.

Перечислим основные функции файловой системы.

1. Идентификация файлов. Связывание имени файла с выделенным ему пространством внешней памяти.

2. Распределение внешней памяти между файлами. Для работы с конкретным файлом пользователю не требуется иметь информацию о местоположении этого файла на внешнем носителе информации. Например, для того чтобы загрузить документ в редактор с жесткого диска, нам не нужно знать, на какой стороне какого магнитного диска, на каком цилиндре и в каком секторе находится данный документ.

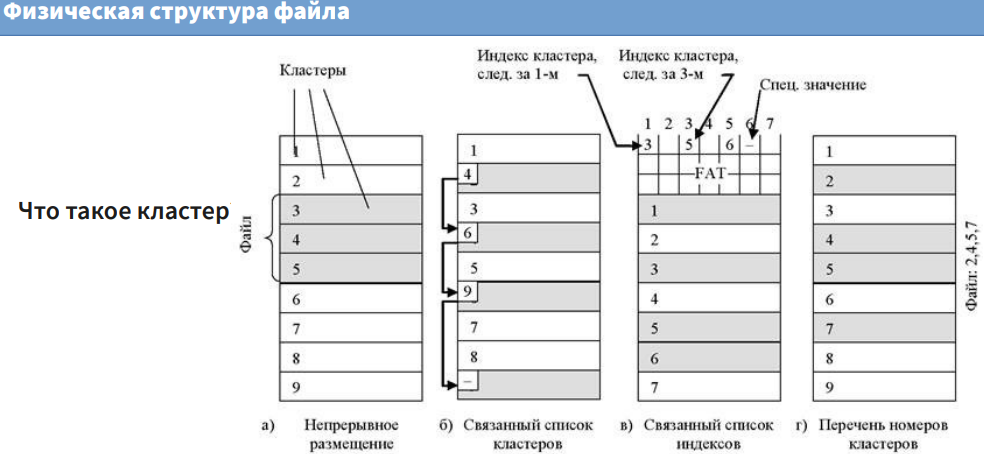
3. Обеспечение надежности и отказоустойчивости. Стоимость информации может во много раз превышать стоимость компьютера.

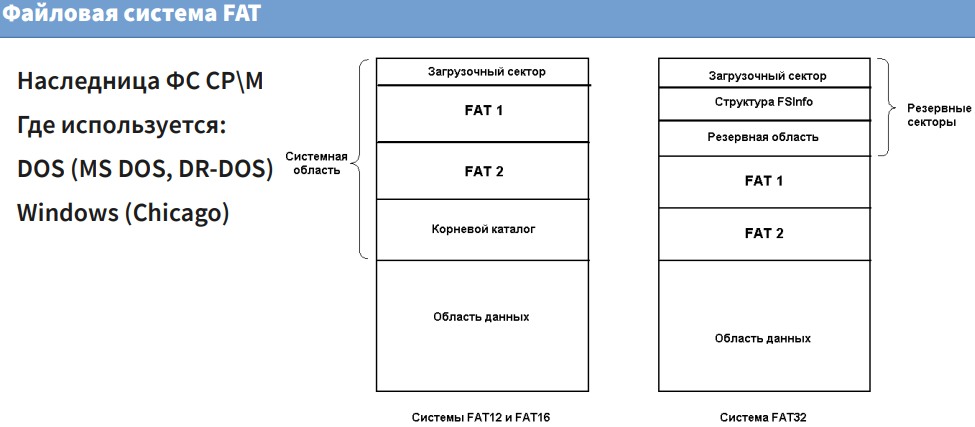
4. Обеспечение защиты от несанкционированного доступа.

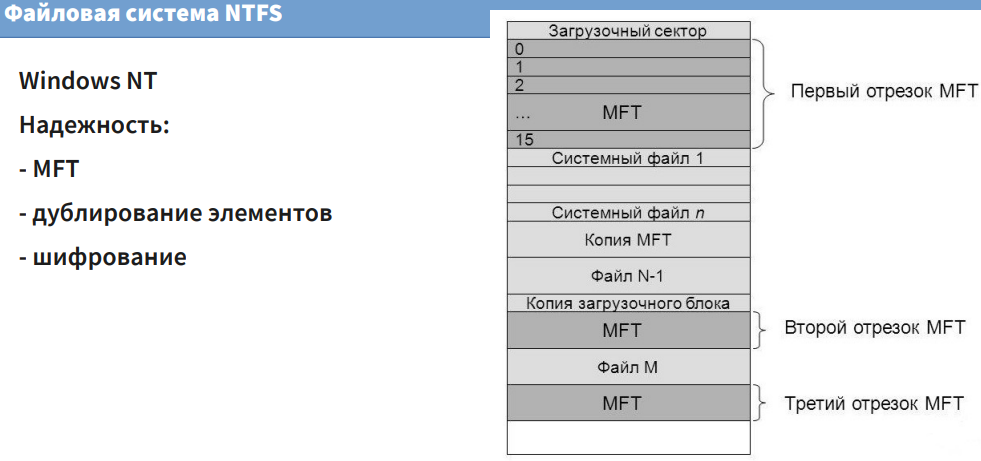
5. Обеспечение совместного доступа к файлам, так чтобы пользователю не приходилось прилагать специальных усилий по обеспечению синхронизации доступа.

6. Обеспечение высокой производительности.

Иногда говорят, что файл - это поименованный набор связанной информации, записанной во вторичную память. Для большинства пользователей файловая система - наиболее видимая часть ОС. Она предоставляет механизм для онлайнового хранения и доступа как к данным, так и к программам для всех пользователей системы. С точки зрения пользователя, файл - единица внешней памяти, то есть данные, записанные на диск, должны быть в составе какого-нибудь файла.







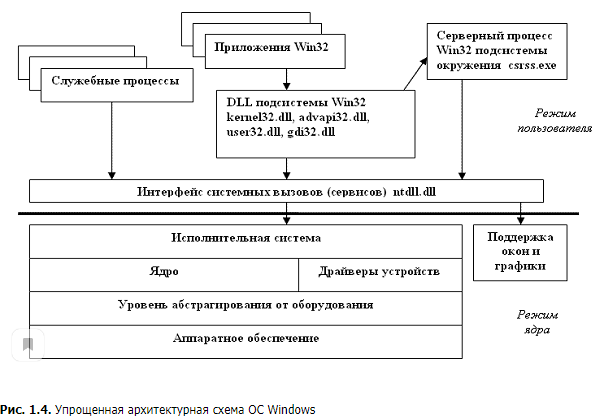
1. **Изобразите схему структурирования переменными страницами и опишите процесс преобразования адресов.**



Максимальный размер страницы L=4 вместо номеров страниц их базовые адреса страниц. Здесь отсутствует взаимооднозначное соответствие между структурированным и непрерывным адресами. Для получения значения непрерывного адреса (Ai, R) необходимо использовать выражение A=Ai+R. По значению непрерывного адреса нельзя сказать к какой странице он принадлежит.

https://pandia.ru/text/78/121/98980-3.php

1. **Изобразите структуру операционной системы на примере Windows и опишите назначение основных модулей.**



ОС Windows состоит из компонентов, работающих в режиме ядра, и компонентов, работающих в режиме пользователя. Несмотря на миграцию системы в сторону монолитного ядра она сохранила некоторую структуру. В схеме, представленной на рис. 1.4, отчетливо просматриваются несколько функциональных уровней, каждый из которых пользуется сервисами более низкого уровня.

Задача уровня абстрагирования от оборудования (hardware abstraction layer, HAL) - скрыть аппаратные различия аппаратных архитектур для потенциального переноса системы с одной платформы на другую. HAL предоставляет выше лежащим уровням аппаратные устройства в абстрактном виде, свободном от индивидуальных особенностей. Это позволяет изолировать ядро, драйверы и исполнительную систему ОС Windows от специфики оборудования (например, от различий между материнскими платами).

Ядром обычно называют все компоненты ОС, работающие в привилегированном режиме работы процессора или в режиме ядра. Корпорация Microsoft называет ядром (kernel) компонент, находящийся в невыгружаемой памяти и содержащий низкоуровневые функции операционной системы, такие, как диспетчеризация прерываний и исключений, планирование потоков и др. Оно также предоставляет набор процедур и базовых объектов, применяемых компонентами высших уровней.

Ядро и HAL являются аппаратно-зависимыми и написаны на языках Си и ассемблера. Верхние уровни написаны на языке Си и являются машинно-независимыми.

Исполнительная система (executive) обеспечивает управление памятью, процессами и потоками, защиту, ввод-вывод и взаимодействие между процессами. Драйверы устройств содержат аппаратно-зависимый код и обеспечивают трансляцию пользовательских вызовов в запросы, специфичные для конкретных устройств. Подсистема поддержки окон и графики реализует функции графического пользовательского интерфейса (GUI), более известные как Win-32-функции модулей USER и GDI

В пространстве пользователя работают разнообразные сервисы (аналоги демонов в Unix), управляемые диспетчером сервисов и решающие системные задачи. Некоторые системные процессы (например, обработка входа в систему) диспетчером сервисов не управляются и называются фиксированными процессами поддержки системы. Пользовательские приложения (user applications) бывают пяти типов: Win32, Windows 3.1, MS-DOS, POSIX и OS/2 1.2. Среду для выполнения пользовательских процессов предоставляют три подсистемы окружения: Win32, POSIX и OS/2. Таким образом, пользовательские приложения не могут вызывать системные вызовы ОС Windows напрямую, а вынуждены обращаться к DLL подсистем (краткое определение dll имеется в приложении).

Основные компоненты ОС Windows реализованы в следующих системных файлах, находящихся в каталоге system32:

* ntoskrnl.exe - исполнительная система и ядро;
* ntdll.dll - внутренние функции поддержки и интерфейсы диспетчера системных сервисов с функциями исполнительной системы;
* hal.dll - уровень абстрагирования от оборудования;
* win32k.sys - часть подсистемы Win32, работающая в режиме ядра;
* kernel32.dll, advapi32.dll, user32.dll, gdi32.dll - основные dll подсистемы Win32.

1. **Изобразите схему сегментно-страничной структуризации и опишите процесс преобразования адресов.**

Этапы:

1-Исходное пространство структурируют исходными страницами.

2-Сегмент рассматривается как уже некоторая непрерывная последовательность номеров страниц. Размер сегмента, количество страниц.

3-Каждый сегмент имеет свой уникальный номер S.

4-В пределах данного сегмента происходит перенумерация страниц, начиная с 0 и в возрастающем.

5-Сегменту, названному базовый адрес As0. В итоге адрес указывается с помощью четырех координат (S, As0, R', R).



Базовый адрес в составе сегмента определяется . L – номер страницы.

Далее, если размер страницы был кратен 2, то к базовому адресу страницы применяют операцию конкатенацию (присоединяют значение смещения).

Основные цели страничной сегментной организации.

Страничная организация ориентирована в первую очередь на удовлетворение нужд системы. Позволяет улучшить использование ОП с уменьшением объёма пересылок между рабочей и архивными адресами. Сегментная организация ориентирована на пользователя, на использование сложных многомодульных программ в мультипрограммной системе.

Механизм преобразования виртуального адреса в физический.

Для каждого сегмента создается своя таблица страниц. Адрес таблицы загружается в специальный регистр процессора в тот момент когда процесс становится активным.

Основные цели страничной и сегментной организации.

Страничная организация ориентирована в первую очередь на удовлетворение нужд системы. Она позволяет улучшить использование оперативной памяти за счет уменьшения объемов пересылок между рабочей и архивной средами.

Сегментная организация в первую очередь ориентирована на пользователя и на исполнение сложных многомодульных программ в мультипрограммном режиме.

1. **Сформулируйте задачи управления виртуальной памятью.**

1-Размещение. В адресном пространстве ОП выбираются страницы и сегменты, на которые будут отображаться некоторые страницы и сегменты виртуального адресного пространства. Сложность в том, что размер виртуального существенно больше линейного адресного пространства ОП. Главная задача уменьшить фрагментацию при размещении.

2-Перемещение. Например, из архивной среды хранения необходимо перенести информацию какой-либо виртуальной страницы и отобразить страницу ОП.

3-Преобразование. Необходимо найти абсолютный адрес в рабочей среде хранения его виртуального адреса в соответствии с функцией преобразования.

4-Замещение. Необходимо выбрать среди страниц адресного пространства кандидата на перераспределение

1. **Опишите виды групп информации дескриптора.**

Процесс – для ОС представляет собой заявку на потребление системных ресурсов. Основной деей процесса является способ управления программами в ходе их выполнения. Процесс создаётся, когда начинается выполнение задания пользователя, и разрушается при его завершении. Процесс, как логическая единица, предполагает два аспекта: выполняет операции, является носителем данных. То есть процессу присущи две части: программа, по которой он будет развиваться в активном состоянии и дескриптор процесса.

Дескриптор процесса – представляет собой информационную структуру, в которой сосредоточена управляющая информация, необходимая для системы планирования и управления процессами.

Контекст процесса – информация о процессе, необходимая непосредственно в активном состоянии.

Группы информации (по функциональному назначению):

* Информация по идентификации – содержит уникальное имя процесса, необходимое для реализации операций управления процессами, как поименованными объектами;
* Информация о ресурсах;
* Информация о состоянии процесса – необходима для определения возможности перехода в следующее состояние;
* Информация о родственных связях – используется для конкретного объекта для правильного окончания выполнения процесса, необходима для указания, какие ресурсы используются совместно, а какие – автономно;
* Информация, необходимая для учёта и планирования процесса – содержит ссылки на средства синхронизации между процессами, а также приоритет или место в соответствующей очереди.

Очередь процессов – дескрипторы отдельных процессов, объединённые в списки.

1. **Приведите характеристики файлов и типы доступа к файлу.**

1-Имя файла. В старых ОС MS-DOS 6.22 и ниже используется формат <8.3> и максимальная длина пути 80 символов. В современных ОС WINDOWS 95 и выше используются длинные имена до 255 символов и длина пути до 260 символов.

2-Расширение файла. ОС должна распознавать стандартный набор расширений.

3-Атрибуты файлов. Специфицируют тип файла, защиту и способ буферизации (пароль для доступа, владелец файла, создатель, признаки только для чтения, скрытый, системный, архивный, временный, текущий размер и др.).

4-Тип файла. Может быть:

* Сегментированный (обеспечивает структуру файла с произвольным доступом и может иметь неограниченный размер).
* Непрерывный (обеспечивает один непрерывный блок и используется для быстрого непосредственного доступа).
* Последовательный (обеспечивает последовательную организацию данных, и файл может расти неограниченно).

По другой классификации файлы бывают:

1-Обычные (текстовые, двоичные).

2-Специальные (например, для операции ввода-вывода. Блоко-ориентированные, байто-ориентированные).

3-Файлы каталоги (справочники, они содержат список файлов и их характеристики).

По типу доступа классифицируются:

1-На чтение.

2-На обновление (модификацию имеющихся записей).

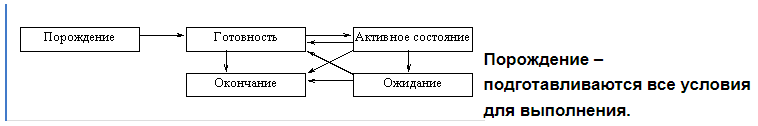
3-На запись (модификацию старых и добавление новых).

4-На удаление.

5-На изменение атрибутов и т. д.

1. **Граф существования процесса. Основные состояния процесса. Условия перехода из одного состояния в другое.**

https://pandia.ru/text/78/121/98980.php



Готовность – предоставляются все ресурсы, но процесс не исполняется, из-за внешних, по отношению к нему, обстоятельств.

Активное состояние – непосредственное использование процессора.

Ожидание – процесс может быть прерван по ряду причин: попытка получения ресурса или отказа от ресурса, порождение, уничтожение или другие действия по отношению к другим процессам, возникновение прерывания (арифметическое переполнение, обращение к защищенной области оперативной памяти и др.), общая необходимость синхронизации между параллельными процессами.

Окончание – нормальное или аварийное завершение работы.

Процесс из состояния бездействия может перейти в состояние готовности в следующих случаях:

1) по команде оператора (пользователя). Имеет место в тех диалоговых операционных системах, где программа может иметь статус задачи (и при этом являться пассивной), а не просто быть исполняемым файлом и только на время исполнения получать статус задачи (как это происходит в большинстве современных ОС для ПК);

2) при выборе из очереди планировщиком (характерно для операционных систем, работающих в пакетном режиме);

3) по вызову из другой задачи (посредством обращения к супервизору один процесс может создать, инициировать, приостановить, остановить, уничтожить другой процесс);

4) по прерыванию от внешнего инициативного устройства [устройство называется инициативным, если по сигналу запроса на прерывание от него должна запускаться некоторая задача] (сигнал о свершении некоторого события может запускать соответствующую задачу);

5) при наступлении запланированного времени запуска программы.

Последние два способа запуска задачи, при которых процесс из состояния бездействия переходит в состояние готовности, характерны для операционных систем реального времени.

1. **Приведите классификацию процессов.**

ПРОЦЕСС – это система действий, реализующая определенную функцию в вычислительной системе и оформленная так, что управляющая программа вычислительной системы может перераспределить ресурс в целях обеспечения мультипрограммирования. Например, программное обеспечение, утилиты и другие системные обрабатывающие программы, а также редактирование какого-либо текста, трансляция, компоновка и использование каких-либо программ.

Управление процессами как в отношении каждого, так и в отношении их совокупности – это функция операционной системы.

В зависимости от ИНТЕРВАЛОВ СУЩЕСТВОВАНИЯ процессы бывают:

* Процессы реального времени – это такой вид процессов, требующие такого планирования, при котором гарантировалось бы окончание процесса до наступления некоторого конкретного момента времени.
* Интерактивные процессы – процессы, время существования которых должно быть не более интервала времени допустимой реакции ЭВМ на запросы пользователя.
* Пакетные процессы – все остальные процессы.

ПО ГЕНЕАЛОГИЧЕСКОМУ признаку:

* Порождающие – вид процессов, задающий требования для порождения других процессов.
* Порожденные – процессы, создаваемые по требованию других процессов.

ПО РЕЗУЛЬТАТИВНОСТИ:

* Эквивалентные – процессы, имеющие одинаковых конечный результат обработки одних и тех же исходных данных по одной и той же или даже различным программам на одном и том же или различных процессах.
* Тождественные – эквивалентные процессы, в которых обработка данных происходит по одной и той же программе, но трассы которых не совпадают.
* Равные – тождественные процессы, трассы которых совпадают.
* Различные – все остальные процессы.

По ДИНАМИЧЕСКОМУ признаку:

* Последовательные – процессы, интервалы которых не пересекаются во времени.
* Параллельные – процессы, существующие одновременно на рассматриваемом промежутке времени.
* Комбинированные – процессы, существующие на рассматриваемом интервале хотя бы в одной точке, но не существующие в других.

По ПРИНАДЛЕЖНОСТИ К ЦЕНТРАЛЬНОМУ ПРОЦЕССОРУ:

* Внутренние (программные) – процессы, развивающиеся на центральном процессоре.
* Внешние – процессы, развитие которых происходит под контролем или управлением операционных систем на процессорах, отличных от центрального.

По ПРИНАДЛЕЖНОСТИ К ОПЕРАЦИОННЫМ СИСТЕМАМ:

* Системные – процессы, при развитии которых исполняется программа из состава операционной системы.
* Пользовательские – процессы, при развитии которых исполняется пользовательская (прикладная) программа.

ПО СВЯЗАННОСТИ:

* Взаимосвязанные – процессы, между которыми, с помощью системы управления процессами, поддерживаются связи какого-либо рода: функциональные, пространственно-временные, управляющие, информационные и т.д.
* Изолированные – процессы со слабыми связями.
* Информационно независимые – взаимосвязанные процессы, при развитии которые используют совместно некоторые ресурсы, но информационно не связанные между собой. Связь между такими процессами может быть либо функциональная, либо пространственно-временная.
* Взаимодействующие – процессы, имеющие информационные связи.
* Конкурирующие – взаимосвязанные процессы, имеющие связи по ресурсам.

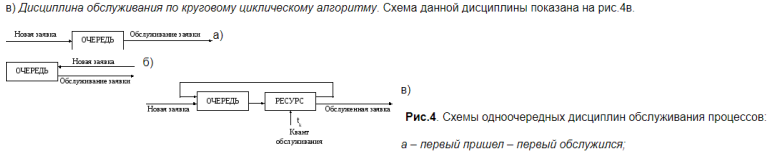
1. **Опишите классические дисциплины обслуживания очереди на исполнение процесса. Примеры алгоритмов планирования.**

Одновременное выполнение нескольких пользовательских программ (мультипрограммирование) неизбежно приводит к появлению очередей, возникающих при обращении процессов к различным ресурсам (центральный процессор, каналы, внешние устройства, наборы данных, модули ОС и т.д.). Очередь формируется из заявок на использование ресурса, поступающих от процессов. Заявки на кратковременное использование ресурса будем называть короткими, а заявки на длительное использование ресурса соответственнодлинными. Не все заявки считаются равноправными, существуют различные системы приоритетов.

Очереди необходимо как формировать, так и обслуживать на основе определенных правил – дисциплин. В зависимости от способа назначения приоритетов, дисциплины формирования очередей делятся на два класса:

• статический– приоритеты назначаются до выполнения задания;

• динамический– приоритеты определяются в процессе выполнения пакета программ.



1. **Приведите структуры организация файла логического уровня.**

50. Логическая организация файла. Файлы с последовательной структурой.

1-Файлы с последовательной структурой. Файл рассматривается как одномерный массив составных элементов. Элемент – называется записью. Длина логических записей может быть как постоянной, так и переменной. Каждая логическая запись характеризуется своим порядковым номером в составе файла. Доступ к файлу последовательный (то есть, после обработки i-ой записи доступна только следующая i+1 запись). Для организации такого доступа достаточно иметь один указатель на текущую запись.

51. Логическая организация файла. Файлы с индексно-последовательной структурой.

2-Файлы с индексно-последовательной структурой. Существует ряд методов, основанных на идентификации записей файла по некоторому ключу. То есть, индивидуальному отличительному признаку. Структура файла усложняется, но сокращается число обращений к диску. Кроме данных дополнительно вводят служебную учётную информацию. Данная структура строится так, что поиск элемента файла проводится с помощью двух методов. В последовательном и прямом порядке. Все записи упорядочиваются по значению ключей. Выделяют группы записей, ключи которых расположены подряд в файле и могут храниться в пределах одной дорожки на диске. Для более быстрого поиска таких групп, строят специальную структуру (индекс). Например, (индекс дорожки) каждый элемент индекса описывает отдельную группу записей. Индекс может содержать максимальный ключ в группе и ссылку на начальную запись в группе. По индексу находят начало первой записи, требуемой группы, а затем внутри группы последовательным алгоритмом имеют требуемую запись.

Недостаток: Проблема расширения файла во время работы с ним. То есть, логика взаимного распространения записей в файле требует упорядоченности по ключам. Для решения проблемы вводят специальную область переполнения, куда заносят записи, динамически вводимые файлы. А затем, из основной области устанавливают ссылки на требуемые элементы области переполнения.

52. Логическая организация файла. Библиотечная структура файлов.

3- Библиотечная структура файлов. Имеется два уровня:

Учётный.

Информационный.

Файл составляется совокупностями последовательных наборов данных, где каждый набор имеет собственное имя в составе данного файла. Такие наборы называют разделами. Расположение разделов не упорядочено и записываются в порядке поступления. Распределение раздела фиксируется в каталоге учётного уровня, а сами элементы каталога расположены в алфавитном порядке.

1. **Вытесняющие алгоритмы планирования процессов.**

Алгоритм использует стратегию, при которой текущий процесс может быть вытеснен другим процессом. Например, после обработки прерывания на выполнение ставится процесс с более высоким приоритетом. при этом вытесненный процесс должен быть повторно обработан планировщиком. Стратегия с вытеснением может чередоваться со стратегией без вытеснения. Например, для каждого процесса вводится два флага: процесс может быть захвачен или нет, и процесс может захватить другой или нет.

1. **Права доступа к файлу. Основные подходы к определению прав доступа.**

Определение права доступа означает определение для пользователя дозволенные операции над файлами. В разных ОС определён свой список операций доступа. Можно выделить следующие операции: создание, уничтожение, открытие, закрытие, поиск, чтение, запись, получение атрибутов, установка атрибутов и др. задаются права в таблице (матрицей прав доступа). В этой таблице указывается следующая информация:

Пользователи могут быть разделены на отдельные категории. При этом подход назначения прав может быть различными для каждой категории, например, в UNIX все пользователи подразделяются на три категории:

1-Владелец файла.

2-Член группы.

3-Все остальные.

Выделяют два подхода к определению прав доступа:

* ·Избирательный доступ (для каждого пользователя и каждого файла сам владелец может определять допустимые операции).
* Мандатный подход (система устанавливает права доступа по отношению к каждому разделу ресурсов в зависимости от того, к какой группе принадлежит пользователь).

1. **Механизм кэширования диска. Достоинства и недостатки.**

В некоторых файловых системах при работе с внешними устройствами используется подсистема буферизации, которая работает по принципу кэш-памяти. Запрос к внешнему устройству, в котором адресация осуществляется блоками, может быть перехвачена подсистемой буферизации. Такая система представляет собой буферный пул (однородных динамически распределяемых блоков ОП одинаковой длины). И комплекс программ управляющих этим пулом. Каждый буфер пула имеет размер равный одному блоку файла.

Механизм работы: При поступлении запроса на чтение некоторого блока подсистема буферизации сначала просматривает свой буферный пул, в случае обнаружения нужного блока, копирует его в буфер, запрашивающего процесса без обращения к внешнему устройству. Если нужный блок не обнаружен, то он считывается и одновременно передается процессу и записывается в буфер подсистемы. При отсутствии свободного буфера на диск вытесняется самая редкая используемая информация. В WINDOWS 95 размер кэша диктуется текущей ситуацией. При интенсивной работе сети ОС увеличивается размер кэша. И/или запуске большого числа приложений автоматически уменьшает размер.

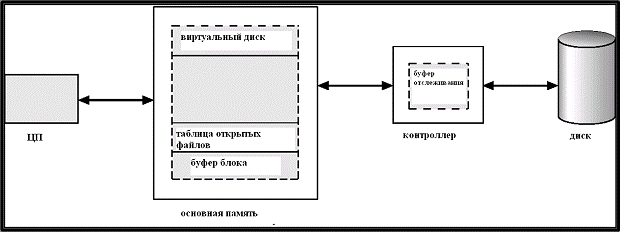
Эффективность использования дисковой памяти зависит от:

* Алгоритмов распределения дисковой памяти и управления директориями;
* Типов данных, хранимых в элементе директории для файла.

Для повышения производительности работы с диском используются следующие методы:

* **Кэширование диска**– использование специальной области основной памяти для часто используемых блоков диска;
* **Освобождение прочитанного (free-behind)**и **опережающее считывание (read-ahead)**– методы оптимизации последовательного доступа к диску, которые заключаются в том, что основная память, в которой хранились копии прочитанных блоков, освобождается, а вместе с очередным блоком файла считываются в основную память и несколько следующих блоков;
* Организация **виртуальных дисков -**улучшение производительности ПК путем выделения области памяти под виртуальный диск (RAM-диск).

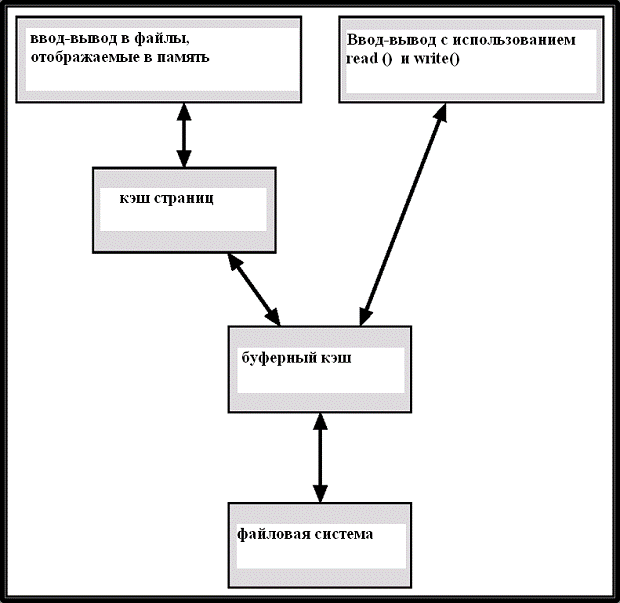
**Буферная кэш-память.**В операционных системах используются различные методы кэширования диска (рис. 20.8).



**Рис. 20.8.** Различные методы размещения кэша для диска.

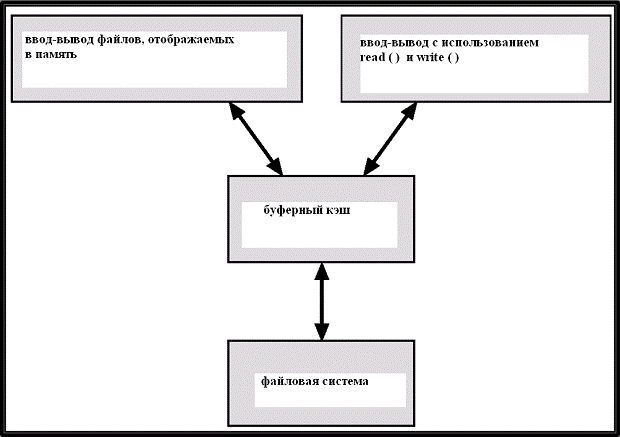
В основной памяти хранятся: виртуальные диски, буфера блоков обрабатываемых файлов, таблицы открытых файлов. В буферной памяти контроллера диска хранятся копии обрабатываемых блоков файла.

Проблема в том, что различные модули кэш-памяти используются не всегда достаточно эффективно и не всегда достаточно согласованы друг с другом. Например, система поддерживает **кэш страниц**, который кэширует страницы, а не блоки файла, используя методы организации виртуальной памяти. Ввод-вывод файлов, отображаемых в память, использует кэш страниц. С другой стороны, обычный ввод-вывод через файловую систему использует кэш буфера (диска). Понятно, что эти различные кэш-модули решают сходные задачи. Не хватает универсализации кэш-памяти. В результате получается усложненная схема, приведенная на рис. 20.9.



**Рис. 20.9.** Ввод-вывод без унифицированной буферной кэш-памяти.

Решение данной проблемы – использование **унифицированной буферной кэш-памяти.**Унифицированная буферная кэш-память использует один и тот же кэш страниц для кэширования и файлов, отображаемых в память, и обычных операций ввода-вывода через файловую систему. Схема ввода-вывода с использованием унифицированной буферной кэш-памяти изображена на рис. 20.10.



**Рис. 20.10.** Ввод-вывод с использованием унифицированной буферной кэш-памяти.

1. **Приведите результаты сравнительного анализа NTFS и ReFS.**

ReFS (Resilient file system, дословно — устройчивая файловая система) — новая ФС, изначально появилась в Windows Server 2012, а начиная с Windows 10 Creators Update в нее можно отформатировать любой несистемный накопитель.

Плюсов у ReFS в сравнении с NTFS много:

Максимальная длина пути к файлу 32768 символов (в NTFS — 255).

Максимальный размер тома 262144 экзабайта (в NTFS — 16).

Более быстрый поиск по системе, а также лучшая надежность благодаря использованию B+ деревьев для хранения данных.

Отсутствие поддержки имен файлов DOS (с тильдой ~, были в NTFS для совместимости со старыми программами, сейчас это не актуально)

Улучшенная защита от потери данных: на дисках хранятся контрольные суммы для метаданных и файлов. При любой транзакции файловой системы (то есть при перемещении файлов) старые метаданные не перезаписываются, а записываются в новый блок и организуются в пачки. Для всех метаданных в ReFS используются 64-битные контрольные суммы, хранящиеся независимо, и если файлы при транзакции были повреждены — файловая система восстановит их сама, без использования CHKDSK.

Windows не может загрузиться из файловой системы ReFS, требуя при этом NTFS.

В ReFS отсутствуют многие технологии, доступные в NTFS. Например, сжатие и шифрование файловой системы, жесткие ссылки, расширенные атрибуты, дедупликацию данных и дисковые квоты. При этом в отличие от NTFS ReFS поддерживает технологию полного шифрования данных — BitLocker.

1. **Задачи и свойства распределенных файловых систем.**

Основные вопросы, решаемые распределёнными ФС:

1-Вопрос обеспечения пространства имен используется два варианта:

* Каждый клиент использует один и тот же путь для доступа к определенному файлу.
* У каждого клиента свое пространство имен и реализуется это путем монтирования разделяемых под деревья к произвольным ключам в иерархии файлов.

2-Вопрос определения вектора состояний:

* Сервер обеспечивает хранение информации об операциях клиента между запросами. Такая информация используется для корректного выполнения следующих запросов, например, запоминать, какие файлы клиент открыл, а также смещение внутри файла и другую информацию.
* Достоинство: серверы работают быстрее. Недостаток: нужны дополнительные ресурсы.

3-Сервер без сохранения состояний. Он более простой в разработке и реализации, но имеет меньшую производительность.

4-Семантики распределения файлов. Этот вопрос важен, когда несколько клиентов обращаются к одному файлу.

Общие требования ОС: изменение одним клиентом должны быть видны другим клиентам.

Подходы:

1-Сессионная семантика – подход «открыть-закрыть», то есть только после закрытия файла одним процессом, другие процессы могут видеть изменения. Особенность: последний закрытый вариант считается окончательным.

2-Должен пройти определённый интервал времени, после которого изменения попадут к другим.

3-Каждая операция немедленно становится видна другим (такая семантика в UNIX).

4-Транзакции. Используется принцип «всё или ничего». Один процесс открывает файл (объекты), он может начать транзакцию с другими процессами, то есть, они тоже могут выполнять операции. Инициатор может объявить, что он завершает работу и если с ним все согласны, то результат фиксируется.

5-Подход неизменяемые файлы. Всегда доступны 2 операции: создать и прочитать. Файл, например, модифицируется, а затем старый заменяется целиком.

6-Методы удалённого доступа.

Свойства распределённых ФС:

В основе распределений ФС лежит модель клиент-сервер. В данном случае под клиентом подразумевается машина, которая обращается к некоторому файлу, а под сервером понимается машина, хранящая файлы и обеспечивающая доступ к ним. Распределенные ФС имеют ряд важных свойств, причём каждая конкретная система может обладать всеми или частью этих свойств (создает основу для сравнения различных архитектур между собой).

Свойства:

1-Сетевая прозрачность. Клиенты должны иметь возможность обращаться к удаленным файлам, пользуясь теми же самыми операциями, что и для доступа к локальным данным.

2-Прозрачность размещений. Имя файла не должно определять его местоположение в сети.

3-Независимость размещения. Имя файла не должно меняться при изменении его физического местоположения.

4-Мобильность пользователя. Пользователи должны иметь возможность обращаться к разделяемым файлам из любого узла сети.

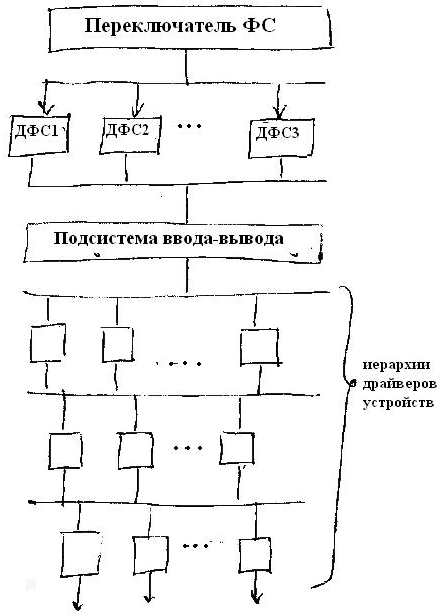
5-Устойчивость к сбоям. Система должна продолжать функционировать при неисправности отдельного компонента сервера или фрагмента сети. Однако это может приводить к уменьшению производительности или к исключению доступа к некоторой части ФС.

6-Масштабируемость. Система должна обладать возможностью масштабирования в случае увеличения нагрузки. Кроме этого должна существовать возможность постепенного наращивания системы, путем добавления отдельных компонентов.

7-Мобильность файлов. Должна быть обеспечена возможность перемещения файлов из одного места в другое в пределах распределения системы.

1. **Опишите архитектуру современной файловой системы.**

Архитектура современной файловой системы такова:



Приложение обращается к такой файловой системе только через переключатель. В WINDOWS 95 такой переключатель называется устанавливаемым диспетчером файловой системы IFS (Installable File System manager). Переключатель преобразует запросы в формат следующего уровня конкретной файловой системы. Каждый диспетчер файловой системы поддерживает определённую организацию файловой системы и позволяет сразу нескольким приложениям выполнять операции с файлами.

DFS – представляет собой расширение сетевого сервиса, позволяющая объединить в единый логический том сетевые ресурсы, причем расположенных в разделах с различными FS.

Подсистема ввода-вывода отвечает за загрузку, инициализацию и управление всеми модулями низких уровней файловой системы. Данная подсистема постоянно присутствует в памяти и организует совместную работу иерархии драйверов устройств. Каждый уровень драйверов устройств, представляет определённый тип (драйвер HDD, драйвер, перехватывающий запросы к блокам, драйверы портов и т. д.)

1. **Опишите механизмы синхронизации и взаимодействия процессов.**

Синхронизация процессов — приведение двух или нескольких процессов к такому их протеканию, когда определённые стадии разных процессов совершаются в определённом порядке, либо одновременно.

Синхронизация необходима в любых случаях, когда параллельно протекающим процессам необходимо взаимодействовать. Для её организации используются средства межпроцессного взаимодействия.

Среди наиболее часто используемых средств — сигналы и сообщения, семафоры и мьютексы, каналы (англ. pipe), совместно используемая память.

Процесс считается свободным, когда его последний поток завершается или заканчивается квант времени.

Понятие свободен применяется и к другим объектам, которые имеют отношение к синхронизации. Объект поток свободен, когда завершается и исполняется поток. Объект события считается свободным, когда поток установлен потоком. Объект семафор считается свободным, когда счетчик семафора доходит до 0. Объект таймер – когда задание время истекает за заданный интервал

1. **События. Семафоры. Мьютексы. Сообщения. Их основное назначение.**

При решении общих задач процессы должны иметь возможность обмениваться данными. Передаваемая последовательность данных называется сообщением.

Синхронизация подразумевает сигнализацию между процессами по определенному протоколу.

Протокол – набор правил и соглашений. Такая операция не зависит от времени и ее принято называть событие. В ОС допускается одновременное возникновение нескольких событий, каждому событию присваивают идентификатор, который называют флагом события. И этот флаг определяет возможность синхронизации. Количество флагов задается при проектировании. С каждым событием связано статическое слово и числовые значения. Если статус меньше 0, то это означает, что событие прекращено из-за ошибки и статическое слово содержит системный код ошибки. Если статус равен 0, то событие продолжается. Если статус больше 0, то это означает удачное завершение.

Простейший вариант синхронизации двух процессов может быть представлен:

А – процесс прародитель.

В – подчиненный процесс.

Процесс А инициализирует процесс В с помощью события 1, а после завершения процесс В посылает статус и значение для события 1. При решении общих задач процессы должны иметь возможность обмениваться данными, то есть взаимодействовать. Передававшую последовательность данных называют сообщением. Обмен сообщениями между процессами имеет сложную структуру, чем просто сигнал. Такая связь между процессами создает новые проблемы, касающиеся адресации процессов структуры планировщиков процесса и др. Асинхронность выполнения процессов усложняет взаимодействие, так как предполагается, что один процесс выполняется со скоростью независящих от других процессов. Принимающий сообщение процесс может быть не готов к приему, чтобы обеспечить взаимодействие используют временной буфер для хранения сообщений, когда принимающий не будет готов к приему. Обмен между процессами может быть разделен на два класса:

1 класс – называется разделяемые совместно используемые переменные (контролер события и семафоры).

Семафоры – это не отрицательные целые переменные, для которых определены две операции:

Операция P – уменьшение семафора на 1. Если это возможно. Если P=0, то процесс вызвавший операцию ждет, пока уменьшение не станет возможным.

Операция V – семафор увеличивается на 1, что предотвращает доступ к семафору других процессов. За исключением того, кто эту операцию выполняет. (Разделяемые переменные).

2 класс - сообщение – обмен информации между процессами имеет 2 ограничения со стороны ресурсов.

Первое ограничение – объем сообщений не должен превышать емкости отведенного буфера.

Второе ограничение – принимающий ресурс не может обрабатывать сообщение быстрее, чем они создаются передающими процессами.

Возможны два подхода при обмене:

Первый подход – процесс может обмениваться только с процессами, имеющими общего прародителя.

Второй - С любыми процессами.

Мью́текс (англ. mutex, от англ. mutual exclusion — «взаимное исключение») — является аналогом одноместного семафора, в программировании необходим для сопоставления синхронно выполняющихся потоков.[1]. Мью́текс представляет собой концепцию программирования, которая используется для решения вопросов многопоточности. Мьютекс отличается от семафора тем, что допускает только один поток в контролируемом участке, заставляя другие потоки, которые пытаются получить доступ к этому разделу ждать, пока первый поток не вышел из этого раздела.

Принимает два значенения:

открыт - поток может войти в свою критическую секцию;

закрыт - поток не может войти в критическую секцию.

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

Цель использования мьютексов — защита данных от повреждения в результате асинхронных изменений (состояние гонки), однако могут порождаться другие проблемы — такие, как взаимная блокировка (клинч). Для описания мьютекса требуется всего один бит, хотя чаще используется целая переменная, у которой 0 означает не блокированное состояние, а все остальные значения соответствуют блокированному состоянию. Значение мьютекса устанавливается двумя процедурами. Если поток собирается войти в критическую область, он вызывает процедуру mutex\_lock. Если мьютекс не заблокирован, запрос выполняется и вызывающий поток может попасть в критическую область[2]. Если mutex закрыт, то поток пытающийся войти в критическую секцию блокируется.

1. **Процесс как объект на высоком уровне абстракции. Атрибуты и сервисы процесса-объекта.**

Процесс представляет собой один из типов объектов, который связан с другими объектами. К другим можно отнести: поток, секция (совместно используемая память), объект файл, объект порт (для передачи данных м-у процессами), объект маркер доступа (закодированный идентификатор с информацией о праве доступа), объект событие, объект семафор и др. объекты.

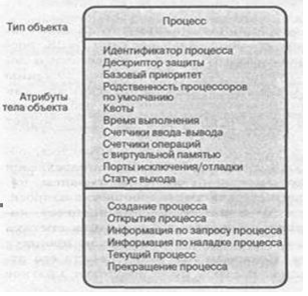
На высоком уровне абстракции процесс состоит из:

1-Из исполнительной программы, которая определяет начальный код и данные процесса.

2-Из закрытого адресного пространства состоит из наборов адресов виртуальной памяти.

3-Из системных ресурсов, таких как семафоров, коммуникационные порты и файлы.

Большинство процессов – это процессы пользовательского режима. В режиме ядра в основном используется код ОС или осуществляется доступ к памяти.



1. **Провести сравнительный анализ ОС Linux и Windows по функционалу систем.**

Ядро

Ядро Linux является монолитным, состоит из одного единственного файла, в случае необходимости расширения функционала дополнительно используют специальные модули.

Общение программ с ядром происходит с помощью системных вызовов. Они стандартизированы, а это значит, что одно и то же ПО без переписывания может функционировать на разных платформах под управлением Linux.

Windows имеет кардинально другой вид ядра. Оно состоит из множества небольших частей библиотек dll, каждая из которых отвечает за свою функцию. Системные вызовы вообще не применяются. В замену этому пользовательские программы обращаются к библиотекам user32.dll, gdi32.dll, kenel32.dll, advapi32.dll, которые вызывают функции из ntdll.dll

Настройки

В Linux все настройки хранятся в обычных файлах, которые расположены в файловой системе. Глобальные файлы настроек находятся в папке /etc/. Они применимы ко всем пользователям, которые используют этот компьютер. Настройки пользовательских программ находятся в скрытых подкаталогах домашнего каталога пользователя.

Windows хранит все настройки приложений, системы и драйверов в специальной базе данных, под названием реестр Windows. Все настройки разделены по ветвях и ключах, а программы могут очень быстро получить к ним доступ.

Доступ

Пользователь Linux имеет доступ к исходному коду ядра и может изменять его в соответствии со своими нуждами. У этого есть свои преимущества, такие как то, что баги в ОС могут быть оперативно найдены и исправлены, а также свои недостатки – разработчики вредоносного ПО могут воспользоваться любой слабостью в системе, если смогут найти ее.

В Windows у пользователей нет свободного доступа к исходному коду ядра

1. **Провести сравнительный анализ Android и Windowsпо свойству надежность. (скорее всего не то, по крайней мере с виндовсом)**

В Linux имеется механизм Out-Of-Memory Killer, который стремится восстановить работоспособность при крайне низком объеме доступной памяти. Ситуация дефицита памяти в современных операционных системах требует пояснения. При страничной организации памяти и подкачке сами приложения сталкиваются со сбоями, связанными с дефицитом памяти, крайне редко. Но ядро все же может попасть в ситуацию, при которой найти необходимые доступные страницы оперативной памяти невозможно не только для нового выделения памяти, но и для замены или подкачки в некотором используемом в данный момент диапазоне адресов.

В ситуации дефицита памяти стандартный механизм его устранения является последней попыткой поиска оперативной памяти, позволяющей ядру продолжить текущую работу. Это делается путем присвоения каждому процессу уровня «вредности» и уничтожения процесса, считающегося самым «вредным». «Вредность» процесса основывается на объеме используемой им оперативной памяти, продолжительности его работы и других факторах, а цель состоит в том, чтобы уничтожить ресурсоемкие процессы, которые, надо надеяться, не играют важной роли.

Android испытывает особую потребность в механизме устранения дефицита памяти. У него нет пространства подкачки, поэтому он попадает в ситуации дефицита памяти значительно чаще прочих: нет способа снижения этого дефицита, кроме выделения чистых страниц оперативной памяти, отображенных из хранилища и недавно использованных. Но даже при этом в Android используется стандартная Linux-конфигурация для выделения памяти в условиях дефицита, то есть разрешение адресному пространству быть выделенным в оперативной памяти без гарантии наличия доступной оперативной памяти для поддержки этого выделения. Выделение памяти в условиях дефицита является особенно важным средством для оптимизации использования памяти, поскольку отображение больших файлов (например, исполняемых) на память с помощью mmap там, где нужно будет лишь загрузить в оперативную память небольшую часть общего объема данных, имеющихся в этом файле, встречается довольно часто.

Центр поддержки. Новое окно для контроля параметров работы системы. С помощью Центра поддержки вы можете управлять параметрами безопасности и обслуживания, а также запускать средства автоматического поиска и устранения проблем.

Безопасность приложений в Android выстраивается вокруг UID-идентификаторов. В Linux каждый процесс работает под конкретным UID, а в Android UID используется для идентификации и защиты барьеров безопасности. Единственный способ взаимодействия процессов заключается в применении механизма IPC, который обычно несет в себе достаточно информации для идентификации UID вызывающего процесса. Binder IPC включает эту информацию явным образом в каждую транзакцию, доставляемую от одного процесса другому, поэтому получатель IPC может запросто затребовать UID вызывающего процесса.

• Новая технология установки системы. Установка Windows 7 выполняется из заранее подготовленных образов, что ускоряет сам процесс установки и снижает вероятность появления ошибок. Для установки любой версии Windows 7 используется единый дистрибутив.

• Средства архивации и восстановления. В Windows 7 имеются удобные средства для автоматической архивации файлов, создания архивных образов разделов жесткого диска, восстановления предыдущих версий файлов и отката к предыдущему состоянию системы.

1. **Провести сравнительный анализ Linux и Windowsпо свойству защиты.**

В операционной системе Linux и модель защиты, и ее реализация по существу точно такие же, как и у большинства традиционных систем UNIX.

Сообщество пользователей операционной системы Linux состоит из зарегистрированных пользователей, каждый из которых имеет уникальный UID (User ID — идентификатор пользователя ). UID представляет собой целое число в пределах от 0 до 65 535. Идентификатором владельца помечаются файлы, процессы и другие ресурсы. По умолчанию владельцем файла является пользователь, создавший этот файл (хотя есть способ сменить владельца). Пользователи могут организовываться в группы, которые также нумеруются 16-битными целыми числами, называемыми GID (Group ID — идентификатор группы ). Назначение пользователя в группу выполняется вручную системным администратором и заключается в создании нескольких записей (в системной базе данных), в которых содержится информация о том, какой пользователь к какой группе принадлежит. Пользователь может одновременно принадлежать к нескольким группам.

Виндовс:

Деление NTOS на ядро и исполнительную систему отражает общность NT с системой VAX/VMS. Операционная система VMS, которая также была разработана Катлером, имела четыре аппаратно обеспечиваемых уровня: пользовательский, супервизора, исполнительный и уровень ядра (в соответствии с четырьмя режимами защиты, обеспечивавшимися архитектурой процессора VAX). Процессор Intel также поддерживает четыре кольца защиты, однако в некоторых процессорах (для которых первоначально разрабатывалась NT) этого не было, поэтому уровни ядра и исполнительный представляют собой программную абстракцию и такие функции, которые VMS предоставляет в режиме супервизора (например, очередь печати), в NT реализованы как службы пользовательского режима.

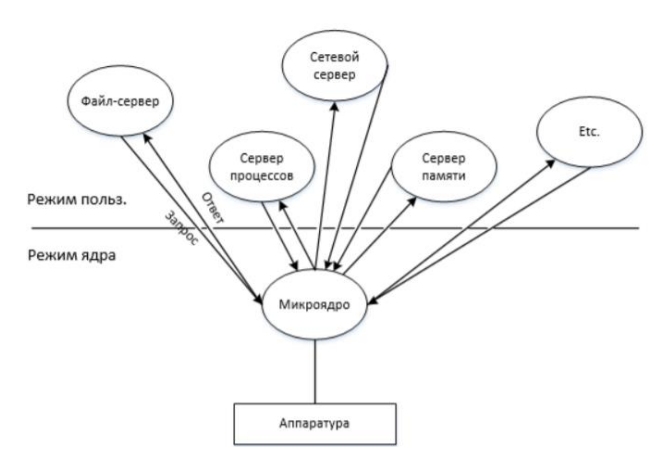
Самым нижним уровнем программного обеспечения является гипервизор, который в Windows называется Hyper-V. Гипервизор является дополнительным средством (на рис. 11.4 не показано). Он доступен на многих версиях Windows, включая профессиональные клиентские версии для настольных систем. Гипервизор перехватывает многие привилегированные операции, выполняемые ядром, и эмулирует их таким образом, чтобы позволить на одной и той же машине одновременно работать нескольким операционным системам. Каждая операционная система работает на собственной виртуальной машине, которая в Windows называется разделом (partition). Гипервизор использует возможности в архитектуре оборудования для защиты физической памяти и обеспечения изолированности разделов друг от друга.

1. **Постройте обобщенную схему двухуровневой операционной системы.**

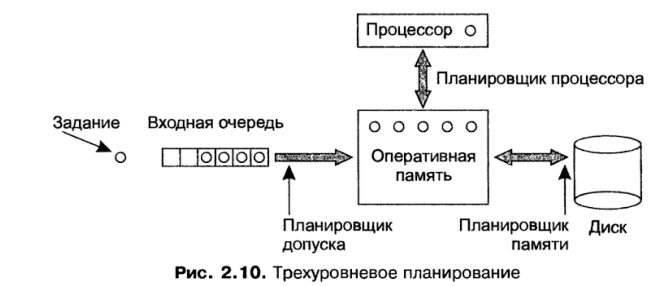
Система является двухуровневой, если в ней реализовано два уровня привилегий - режим ядра (привилегированный) и режим пользователя

В микроядерных архитектурах, вертикальное распределение функций заменяется на горизонтальное. Т.е. компоненты взаимодействуют непосредственно с ядром и используют средства микроядра для обмена сообщениями. Микроядро проверяет «законность» сообщений, пересылает их между компонентами, а также обеспечивает доступ к аппаратуре. Такой подход позволяет использовать микроядерные ОС в распределенных системах. Объясняется это тем, что микроядру безразлично – поступило ли сообщение от локального компьютера или удаленного

Недостатки: пересылка сообщений производится медленнее, чем обычные вызовы функций (как в предыдущих архитектурах), поэтому одной из задач является оптимизация пересылки сообщений

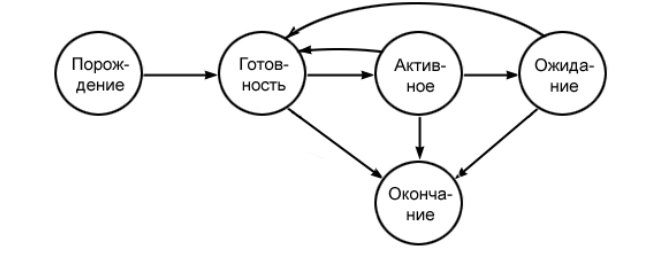


1. **Постройте обобщенную схему алгоритма работы планировщика памяти.**



После поступления в систему новые задания сначала помещаются в очередь, хранимую на жестком диске. Далее планировщик допуска решает, какие задания принять в систему. Остальные ожидают во входной очереди, пока не будут выбраны планировщиком. Как правило, контроль допуска обеспечивает желательное сочетание заданий, ограниченных возможностями процессора и ввода-вывода. Возможен и другой подход — короткие задания выбираются быстрее, чем длительные. Планировщик допуска может выбирать задания из очереди по собственному усмотрению, независимо от того, какие из них пришли раньше, а какие — позже.

1. **Провести сравнительный анализ по основным способам организации вычислительного процесса операционных систем Linux и Windows.**
2. **Постройте граф существования процесса для ОС общего назначения.**



*порождение* — подготавливаются условия для первого исполнения на процессоре; *активное состояние*, или состояние “Счет” — программа исполняется на процессоре; *ожидание* — программа не исполняется на процессоре по причине занятости какого-либо требуемого ресурса;

*готовность* — программа не исполняется, но для исполнения предоставлены все необходимые в текущий момент ресурсы, кроме центрального процессора;

*окончание* — нормальное или аварийное окончание исполнения программы, после которого процессор и другие ресурсы ей не предоставляются.

Порождение->готовность - получение всех необходимых ресурсов, кроме процессора Готовность->Активное получение процессора

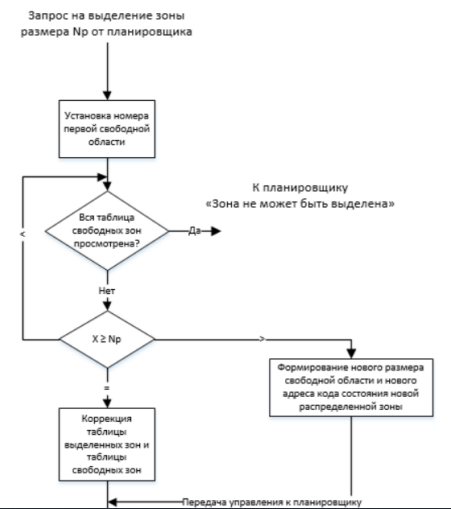
Активное->Готовность окончание времени работы на процессоре, но не завершение процесса

Активное->Ожидание Потеря какого-либо ресурса (например области памяти) Готовность->Ожидание (скрытая стрелка) Потеря какого-либо ресурса

Ожидание->Готовность Получение необходимых ресурсов

…->Окончание - нормальное или аварийное завершение работы

1. **Постройте схему алгоритма выделения ресурсов оперативной памяти для процесса верхнего уровня.**



Основная идея работы системы:

1) Если памяти достаточно для выполнения задач, то цель – повысить эффективность доступа к данным;

2) Если памяти недостаточно, то цель – эффективная загрузка. В случае дефицита памяти используется подход, при котором пользователь работает с памятью не на физическом, а на логическом (виртуальном) уровне

Если ОС двухуровневая, то память на верхнем распределяется статически, а на нижнем – динамически.

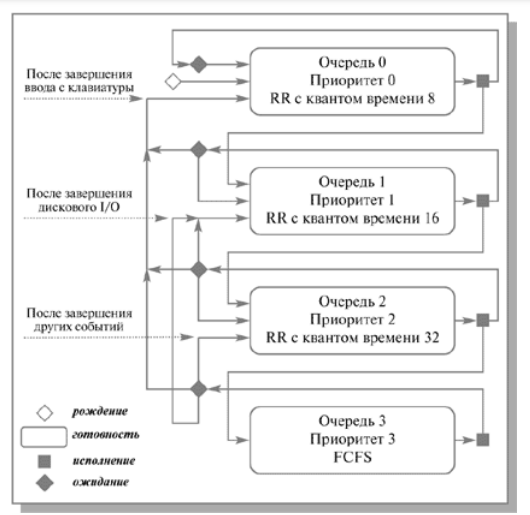
На каждом уровне решаются три задачи:

1) Выделение памяти;

2) Учет памяти;

3) Возврат памяти.

1. **Постройте многоочередную схему управления процессами.**



Многоуровневые очереди с обратной связью представляют собой наиболее общий подход к планированию процессов из числа подходов, рассмотренных нами. Они наиболее трудны в реализации, но в то же время обладают наибольшей гибкостью. Понятно, что существует много других разновидностей такого способа планирования, помимо варианта, приведенного выше. Для полного описания их конкретного воплощения необходимо указать:

Количество очередей для процессов, находящихся в состоянии готовность. Алгоритм планирования, действующий между очередями.

Алгоритмы планирования, действующие внутри очередей.

Правила помещения родившегося процесса в одну из очередей.

Правила перевода процессов из одной очереди в другую.

Изменяя какой-либо из перечисленных пунктов, мы можем существенно менять поведение вычислительной системы