Лекции по дисциплине «Операционные системы»

Преподаватель: Михелев Владимир Михайлович

(6 лаб, Экзамен, Автомат первым трем)

8 сентября

Основные понятия.

Операционная система — это программное обеспечение, которое с одной стороны выступает, как интерфейс между аппаратурой компьютера и пользователем с его задачами, а с другой стороны предназначено для организации эффективного использования ресурсов этого компьютера и организации надежных вычислений.

ОС изолирует софт от аппаратуры.

Эволюция операционных систем:

1) Ламповые компьютеры. Была сформирована модель Фон-Неймана
2) Компьютеры на транзисторах. Первый язык программирования Fortran. Появились операторы. Первые системы пакетной обработки данных.
3) Компьютеры на интегральных микросхемах. (IBM 360/370) 65-80 годы. Были получены основные концепции, которых придерживаются сегодняшние ОС.
    1) Было реализовано мультипрограммирование (появилась возможность выполнения нескольких программ)
    2) Мультипроцессирование (появлялись компьютеры с несколькими процессорами)
    3) Поддержка многотерминального, многопользовательского режима работы
    4) Виртуальная память
    5) Файловая система
    6) Разграничение доступа
    7) Сетевая работа
4) CTSS - система совместного распределения времени. Multics - Мультипроцессорная вычислительная служба
5) Компьютеры на БИС (Больших интегральных схемах). Процессор: Intel

WinNT, WinNT 5.0, WinNT 5.1 2003 (XP), …

Классификация ОС:

1) Однопользовательские
2) Многопользовательские

2) По доступу (типу организации)
    1) Пакетные
       Необходимость оптимального использования дорогостоящих вычислительных ресурсов привела к появлению концепции «пакетного режима» исполнения программ. Пакетный режим предполагает наличие очереди программ на исполнение, причём система может обеспечивать загрузку программы с внешних носителей данных в оперативную память, не дожидаясь завершения исполнения предыдущей программы, что позволяет избежать простоя процессора.
    2) Системы распределения времени
       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.
   3) Системы реального времени (QNX)
3) По количеству решаемых задач
   1) Однозадачная
   2) Многозадачная
4) По виду интерфейса
   1) Символьный
   2) Графический
5) По назначению
   1) Общего назначения
   2) Спец назначения
   3) Мультипроцессорные системы

Функции ОС:
1) Представление пользователю вместо реальной аппаратуры компьютера некой расширенной виртуальной машины. (В книгах «ОС, как виртуальная машина»)
2) Контроль и управление ресурсами компьютера.

Операционная система как виртуальная машина
При разработке ОС широко применяется абстрагирование, которое является важным методом упрощения и позволяет сконцентрироваться на взаимодействии высокоуровневых компонентов системы, игнорируя детали их реализации. В этом смысле ОС представляет собой интерфейс между пользователем и компьютером.

Архитектура большинства компьютеров на уровне машинных команд очень неудобна для использования прикладными программами. Например, работа с диском предполагает знание внутреннего устройства его электронного компонента – контроллера для ввода команд вращения диска, поиска и форматирования дорожек, чтения и записи секторов и т. д. Ясно, что средний программист не в состоянии учитывать все особенности работы оборудования (в современной терминологии – заниматься разработкой драйверов устройств), а должен иметь простую высокоуровневую абстракцию, скажем, представляя информационное пространство диска как набор файлов. Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать. Это концептуально проще, чем заботиться о деталях перемещения головок дисков или организации работы мотора. Аналогичным образом, с помощью простых и ясных абстракций, скрываются от программиста все ненужные подробности организации прерываний, работы таймера, управления памятью и т. д. Более того, на современных вычислительных комплексах можно создать иллюзию неограниченного размера оперативной памяти и числа процессоров. Всем этим занимается операционная система. Таким образом, операционная система представляется пользователю виртуальной машиной, с которой проще иметь дело, чем непосредственно с оборудованием компьютера.

ОС как система управления ресурсами

Ресурс — совокупность различных составляющих компьютера (ОЗУ, процессор, таймер, наборы данных и т.д.
Процесс — программа в стадии выполнения.
Программа — статический объект представляющий собой файл с кодами и данными.
ОС распределяет ресурсы между процессами и должна управлять этими ресурсами с целью их наиболее эффективного использования.

ОС управляя ресурсами решает следующие задачи:
1) Планирование ресурсов — ранжирование
2) Удовлетворение запросов на эти ресурсы
3) Отслеживание состояния ресурсов с учетом их использования
4) Разрешение конфликтов

Операционная система предназначена для управления всеми частями весьма сложной архитектуры компьютера. Представим, к примеру, что произойдет, если несколько программ, работающих на одном компьютере, будут пытаться одновременно осуществлять вывод на принтер. Мы получили бы мешанину строчек и страниц, выведенных различными программами.
Операционная система предотвращает такого рода хаос за счет буферизации информации, предназначенной для печати, на диске и организации очереди на печать.
Для многопользовательских компьютеров необходимость управления ресурсами и их защиты еще более очевидна. Следовательно, операционная система, как менеджер ресурсов, осуществляет упорядоченное и контролируемое распределение процессоров, памяти и других ресурсов между различными программами.

Функциональные компоненты ОС:
1) Подсистема управления процессами
2) Подсистема управления памятью
3) Подсистема управления файлами и внешними устройствами
4) Пользовательский интерфейс
5) Защита
6) Система администрирования

Управление процессами
Для каждого вновь создаваемого процесса, ОС генерирует **контекст процесса.** Структура, в которой содержатся данные о потребностях процесса в ресурсах. Чтобы процесс был выполнен ОС должна назначить этому процессу область ОП, в которой будут размещены коды и данные процесса, а также предоставить ему процессорное время.

Управление памятью
Управление памятью включает в себя распределение между всеми существующими в ОС процессами, а также загрузку их кода и данных в отведенные области памяти, а также защиту областей памяти каждого процесса.

Управление файлами
С помощью драйверов.

Интерфейс прикладного программирования (API)
Возможности операционной системы доступны прикладному программисту в виде набора функций, называющегося интерфейсом прикладного программирования (Application Programming Interface, API). От конечного пользователяэти функции скрыты за оболочкой алфавитно-цифрового или графического пользовательского интерфейса.

Для разработчиков приложений все особенности конкретной операционной системы представлены особенностями ее API. Поэтому операционные системы с различной внутренней организацией, но с одинаковым набором функций API кажутся им одной и той же ОС, что упрощает стандартизацию операционных систем и обеспечивает переносимость приложений между внутренне различными ОС, соответствующими определенному стандарту на API. Например, следование общим стандартам API UNIX, одним из которых является стандарт Posix, позволяет говорить о некоторой обобщенной операционной системе UNIX, хотя многочисленные версии этой ОС от разных производителей иногда существенно отличаются внутренней организацией.

Приложения выполняют обращения к функциям API с помощью системных вызовов. Способ, которым приложение получает услуги операционной системы, очень похож на вызов подпрограмм. Информация, нужная ОС и состоящая обычно из идентификатора команды и данных, помещается в определенное место памяти, в регистры и/или стек. Затем управление передается операционной системе, которая выполняет требуемую функцию и возвращает результаты через память, регистры или стеки. Если операция проведена неуспешно, то результат включает индикацию ошибки.

Требования к современным ОС

Должна обеспечивать расширяемость, переносимость, совместимость, надежность, производительность.

Архитектура ОС

1) **Монолитная**
   Каждый компонент такой ОС включен в ядро и может непосредственно взаимодействовать с другим компонентом этой ОС. Делает ОС намного эффективной, однако определение источников сбоев и ошибок этой ОС представляет собой трудную задачу. Более того, когда ОС выполняется с неограниченными правами доступа к ресурсам компьютера, то эти ОС особенно уязвимы для вредоносного кода.
2) **Многоуровневая**
   Основана на организации компонентов ОС по уровням, причем каждый уровень взаимодействует только с соседним уровнем расположенным над или под ним.

   Однако, многоуровневая организация требует, чтобы запросы на обслуживание пользовательского процесса проходили через несколько уровней, прежде чем он будет удовлетворен, что снижает производительность такой ОС.

3) **Построенная на микро ядре.**
   Представляет собой малый набор услуг, который включается в ядро. Обычно в перечень этих услуг входит: низкоуровневое управление памятью, межпроцессорное взаимодействие, синхронизация процессов. Большая часть компонентов выполняется вне ядра с более высокими привилегиями (управление процессами, работа в сети, файловая система, управление внешними устройствами). Поэтому сбои с этими модулями обычно не приводят к краху ОС.

Архитектуру ОС обычно разделяют на две части:

1) Ядро — модули выполняющие основные функции ОС
2) Вспомогательные модули

Ядро — сердцевина ОС, без ядра ОС не дееспособна (не может выполнить ни одну из своих функций)

Режим ядра имеет неограниченный доступ к системной памяти и внешним устройствам. Ядро системы NT называют гибридным ядром или макроядром. Архитектура включает в себя само ядро, уровень аппаратных абстракций (HAL), драйверы и ряд служб (Executives), которые работают в режиме ядра (Kernel-mode drivers) или в пользовательском режиме (User-mode drivers).

Пользовательский режим Windows NT состоит из подсистем, передающих запросы ввода-вывода соответствующему драйверу режима ядра посредством менеджера ввода-вывода. Есть две подсистемы на уровне пользователя: подсистема окружения (запускает приложения, написанные для разных операционных систем) и интегрированная подсистема (управляет особыми системными функциями от имени подсистемы окружения). Режим ядра имеет полный доступ к аппаратной части и системным ресурсам компьютера. И также предотвращает доступ к критическим зонам системы со стороны пользовательских служб и приложений.

Многоуровневая структура ОС

Аппаратура — Ядро — Вспомогательные модули — все остальное
Ядро — сложный многофункциональный комплекс и так же обычно имеет многослойную структуру. Выделяют 5 базовых слоев в ядре.

1) Средства аппаратной поддержки ОС (Система прерываний, таймер, средства защиты памяти)
2) Машинно-независимые компоненты ОС (Те модули, которые отражают специфику конкретной аппаратуры компьютера и этот слой экранирует следующие слои ОС от особенностей аппаратуры)
3) Базовый механизм ядра (Диспетчеризация прерываний, перемещение страниц из памяти на диск)
4) Менеджеры ресурсов реализующие стратегические задачи по управлению основными ресурсами компьютера
5) Интерфейс системных вызовов (Взаимодействует с приложениями, системными утилитами, благодаря АПИ предоставляет доступ к определенным характеристикам/ ресурсам ОС в удобном виде)

Архитектура Win NT (New Technologies 3.11 - 93 г; 4.0 - 96 г; 5.0 - 99 г; 5.1 - 2002 г; 6.0 - 2006; 6.1; 6.2)

К службам уровня Hal.dll относятся:

1) Доступ к регистрам устройств
2) Обработка прерываний
3) Операция DMA (Direct Memory access)
4) Управление таймерами
5) Интерфейс BIOS (Base Input Output System)
6) Доступ к CMOS

Назначение этого уровня в том, чтобы сделать всю ОС независимой от аппаратуры, чтобы ее можно было переносить на другие платформы. Над ядром и драйверами находится «Исполнительная система», в которую входят 10 базовых менеджеров:

1) Ввода/вывода
2) Объектов
3) Потоков
4) Памяти (реализует виртуальную память со страничной подкачкой)
5) Безопасности
6) Кэша (хранит в памяти блоки жесткого диска, которые использовались последнее время)
7) Plug and play (Уведомляет о новых появившихся устройствах)

8) Энергопотребления
9) Конфигурации (Системный реестр)
10) Процедур
11) Интерфейс графических устройств (GDI, позволяет пользовательским программам выводить данные на монитор и принтер)

Системные службы

Предоставление доступа и интерфейсов к исполнительным системам.

Режим пользователя имеет 3 типа компонентов:

1) Динамические библиотеки
2) Системные процессы
   1) SMSS (диспетчер сеансов)
   2) WINLOCATION (авторизация)
   3) IDLE (время простоя)
3) Подсистемы

Загрузка Windows

- BIOS
  Base Input Output System
  Настройки BIOS хранятся в CMOS (микросхема динамической памяти) объемом 256 байт

- MBR
  Master boot record - главная загрузочная запись)(0 цилиндр, 0 сектор, 0 поверхность
  Если загрузка производится с жёсткого диска, то система считывает первые **512 байт Master Boot Record (MBR)** и передаёт ему управление.
  Содержит исполняемые команды, копирует ntldr в корневой каталог загрузочного диска, а также копирует boot.ini

- ntldr
  NT Loader - загрузчик операционных систем windows NT.
  Переводит процессор в защищенный режим, затем запускает загрузчик ОС, который считывает конфигурацию из boot.ini

- boot.ini
  Возложена функция управления содержимым меню выбора операционной системы во время загрузки компьютера и задание параметров ее дальнейшего функционирования.
  Именно тут записано место расположение NToskrnl, hal.dll

- Запуск NTOSKRNL и загрузка HAL
  Ntoskrnl — это главный файл ядра Windows и исполнительных подсистем. Он содержит Executive, Kernel, Cache Manager, Memory Manager, Scheduler, Security Reference Monitor и другие. Именно Ntoskrnl приводит в действие Windows.
  Для работы Ntoskrnl необходим файл **hal.dll**, который содержит код, позволяющий оборудованию взаимодействовать с операционной системой

- SMSS (Session Manager Subsystem)
  Сеансовый менеджер подсистем настраивает пользовательскую среду. Система выполняет сверку с реестром, для того чтобы иметь возможность начать загрузку оставшихся драйверов и программного обеспечения, которые необходимо добавить.
  Ядро операционной системы также загружает файлы kernel32.dll, gdi32.dll и user32.dll, которые обеспечивают программное обеспечение пользователя доступом к Win32 API.
  запуск процесса регистрации в системе WinLogon

выполняет важные функции инициализации, такие как создание, переменных окружения системы; задание имен устройств MS DOS, например, LPT1 и COM1; загрузка той части подсистемы Win32, которая относится к режиму ядра;.

- CSRSS.exe
  Client/Server Runtime Subsystem - в основном отвечает за обработку консоли в Win32 и графический интерфейс выключения ОС. csrss.exe также включает в себя функциональность менеджера окон, то есть обрабатывает входящие события, такие как нажатие клавиш клавиатуры и мыши, и передает их на обработку соответствующим приложениям. Каждое приложение само производит перерисовку окон в ответ на эти сообщения.

- WINLogon
  которая выводит на экран диалоговое окно регистрации пользователя и загружает процесс Local Security Authority (lsass.exe).

- lsass.exe
  Сервер проверки подлинности локальной системы безопасности (англ. **Local Security Authority** Subsystem Service, LSASS). отвечающей за авторизацию локальных пользователей отдельного компьютера.

- services.exe
  Данный процесс является диспетчером управления службами и отвечает за запуск, остановку и взаимодействие с системными процессами.

- explorer .exe

Процессы и потоки

Архитектуры (симметричная (все вычислители одинаковые), ассиметричная (1 главный)). Все функционирующее на компьютере ПО и включая саму ОС представляют в виде набора процессов. Процессом является выполняемая программа включающая значения счетчика команд, регистров и данных. В процессе работы компьютера процессор должен переключаться с одного процесса на другой (многозадачность).

Выполнение процессов:
1) тетрадь
2) тетрадь

1 физический счетчик команд.
MPI (Message person interface)—позволяет передавать сообщения между процессами выполняющими одну задачу.

Многопоточность — способность ОС поддерживать выполнение несколько потоков в рамках одного процесса. Все современные ОС многопоточны.

В многопоточной системе ресурсы распределятся определенным образом. **Процесс** рассматривается, как единица распределения ресурсов в памяти и защиты.
В рамках процесса может находиться один или несколько потоков, каждый из которых обладает характеристиками:

1) Состояние выполнения потока
2) Сохраненный контекст не выполняющегося потока
3) Стек выполнения
4) Статическая память выделенная потоку для локальных переменных
5) Доступ к памяти и ресурсам процесса, которому этот поток принадлежит

В однопоточной ОС фото 3
В многопоточном модуле фото 4

Когда мы строим многопоточную систему все потоки делят ресурсы процесса.

Основные преимущества потоков

1) Создание нового потока в уже существующем процессе занимает меньше времени, чем создание совершенно нового процесса
2) Поток можно завершить намного быстрее, чем процесс
3) Переключение потока в рамках одного и того же процесса происходит намного быстрее
4) Повышается эффективность обмена информацией различными потоками одного и того же процесса за счет использования общей памяти без использования ядра ОС

Состояния потоков

Существует 3 базовых состояния:

1) **Выполнение**
   Активное состояние потока, во время которого поток обладает всеми ресурсами и непосредственно выполняется на процессоре
2) **Ожидание**
   Пассивное состояние, находясь в котором поток заблокирован по своим внутренним причинам (Операция ввода/вывода, получение сообщений из другого потока, освобождение каких-то ресурсов).
3) **Готовность**
   Пассивное состояние, в этом случае поток заблокирован по внешним причинам (ОС). Ждет предоставления кванта времени.

В течении своей жизни поток переходит из одного состояния в другое в соответствии с алгоритмами планирования потоков.

Когда поток создается переходит в состояние **Готовность ->** когда процессорное время будет предоставлено перейдет в состояние **Выполнения ->** может перейти в состояние **Ожидания ->** или **Готовность ->**, либо **Завершится**

Те потоки, которые находятся в очереди, обычно организуются путем объединения в списки описателей отдельных потоков, при этом каждый такой поток содержит указатель на другой описатель.

13 октября

Планирование и диспетчеризация потоков

Переход потока из одного состояния в другое осуществляется в результате планирования и диспетчеризации.
Под планированием понимается работа по определению того, в какой момент времени необходимо прервать выполнение активного потока и предоставить возможность выполнения к какому-то следующему потоку.

Планирование включает 2 простые задачи:

1) Определение момента снятия текущего активного потока
2) Выбор для выполнения потока из очереди готовых

Существует два вида планировщиков:

1) Статический
   Обычно принимает решения по какому-то расписанию, решение принимается не во

время работы самой системы, а заранее, когда заранее известен какой-то набор действий, которые будут выполняться.

2) Динамический
   Должен принимать решения в режиме online (на основе анализа текущих ситуаций). Т.е. сколько потоков, какие возникают, ведь ОС работает в режиме неопределенности, все возникает в случайные моменты времени и так же завершаются. Все зависит от состояния аппаратуры и т.д.

В ОС есть понятия, как **не вытесняющий или корпоративный алгоритм** планирования, основанный на том, что поток выполняется до тех пор, пока он сам не отдаст управление ОС.
В современных ОС используются **вытесняющиеся алгоритмы**, в которых решение о переключении процессора с выполнения одного потока на другой принимает сама ОС.

Эти алгоритмы планирования должны быть такими, чтобы ОС работала как можно эффективнее.

Критерии алгоритмов планирования

Исходя из вида ОС критерии бывают разные.

1) Пакетная обработка
   Максимальное количество задач за единицу времени

2) Разделения времени
   Время отклика (время между введением какой-то команды и получением результата)

3) Реального времени
   Реактивность

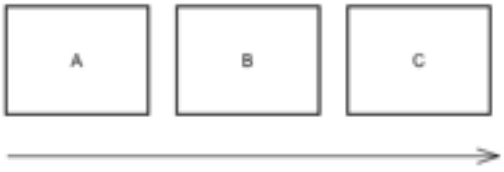
Планирование в системах пакетной обработки:

Критерии:

1) Пропускная способность, максимальное количество задач за единицу времени.
2) Оборотное время (статистически усредненное время от момента получения задачи ОС до ее выполнения).
   Этот показатель характеризует, какой промежуток времени среднестатистический пользователь будет ждать до окончания решения его задачи. (Должен принимать минимальное значение)
3) Коэффициент загрузки процессора

Алгоритм: FCFS (FIFO) (Первый пришел - первый обслужен)

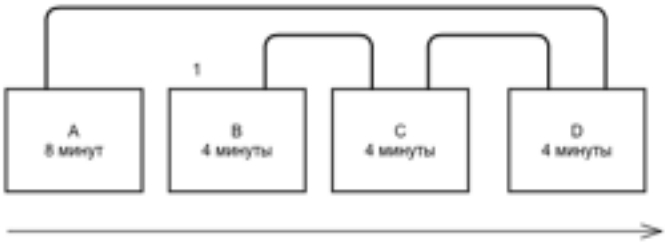
В этом случае тем потокам, которые поступают на выполнение доступ к процессору предоставляется в том порядке, в котором они его запрашивают.



Плюс: легко программируется и используется

Кратчайшая задача — первая

Условие использования: временные рамки использования должны быть заранее известны



$$T_{оборотное} = \frac{(4+8+12+20)}{4} = 11$$

Если бы задачи выполнялись подряд

$$T_{оборотное} = \frac{(8+12+16+20)}{4} = 14$$

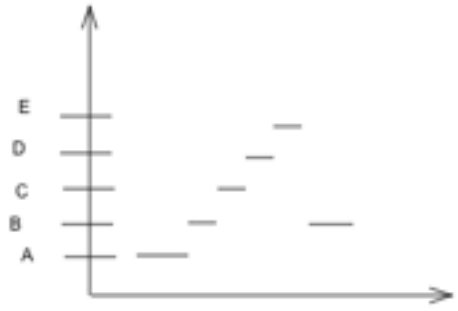
Есть 5 задач: A - 2 мин, B - 4 мин, C -1 мин, D - 1 мин, E - 1 мин
A,B возникают одновременно, а затем через 3 минуты возникают C, D, E

$$T_{оборотное} = \frac{(2+6+4+5+6)}{5} = \frac{23}{5} = 4,6$$

Если выполнять задачи в порядке B, C, D, E, A

$$T_{оборотное} = \frac{(4+2+3+4+9)}{5} = \frac{22}{5} = 4,4$$

Наименьшему оставшемуся времени выполнения
Условие использования: временные рамки использования должны быть заранее известны, время начала может быть любым

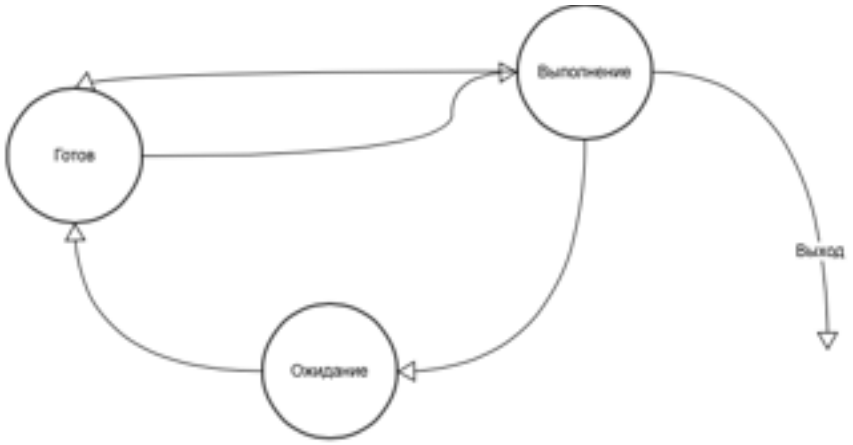


$$T_{оборотное} = \frac{(2+9+1+2+3)}{5} = \frac{17}{5} = 3,4$$

Планирование потоков в системах распределения времени

Цель планирования в системах разделения времени — повышение удобства и эффективности работы пользователя. В системах разделения времени пользователям (или одному пользователю) предоставляется возможность интерактивной работы сразу с несколькими приложениями. ОС принудительно периодически приостанавливает приложения, не дожидаясь, когда они добровольно освободят процессор. Всем приложениям попеременно предоставляется квант процессорного времени, таким образом, что пользователи, запустившие программы на выполнение, получают возможность поддерживать с ними диалог.

Циклическое планирование



Смена активного процесса происходит, если поток завершился и поток покинул систему, либо произошла ошибка

Поток переходит в режим ожидания во время ожидания ввода, исчерпан лимит кванта времени.

Программам, которые не использовали квант времени могут рассчитывать на предоставление еще раз кванта времени.

Приоритетный

Каждому потоку присваивается определенный приоритет и ОС предает готовому к выполнению потоку. Приоритет — число, которое характеризует степень привилегированности потока к …
Чем выше приоритет, тем меньше поток будет находиться в очередях.

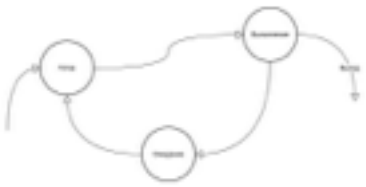
Приоритеты реального времени 16-31, остальные 0-15 потоки с переменным приоритетом. Базовый приоритет и процесса и потока = 8.

Существует две разновидности приоритетного планирования:

1) С относительным приоритетом
   Выполняется пока не выполнится
   В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не освободит процессор, т.е.:
   - процесс покидает систему, поскольку завершился или произошла ошибка, процесс
   - переходит в состояние ОЖИДАНИЕ (т.е. отсутствует переход в состояние

готовность после исчерпания времени работы).

2) С абсолютным приоритетом
Из очереди выбирается поток (очередь выстраивается согласно приоритету), он поступает на выполнение и выполняется до тех пор, пока не выполнится и не завершит свою работу. Если этому потоку необходима операция ввода/вывода переходит в состояние ожидания, и после выполнения ввода попадает в очередь с состоянием готовности.



В системах с абсолютными приоритетами выполнение активного процесса прерывается дополнительно к указанным двум для относительного приоритета еще при одном условии:
- если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности.

3) Справедливое планирование
До сих пор мы предполагали, что каждый процесс управляется независимо от того, кто его хозяин. Поэтому если пользователь 1 создаст 9 процессов, а пользователь 2 — 1 процесс, то с использованием циклического планирования или в случае равных приоритетов пользователю 1 достанется 90 % процессора, а пользователю 2 всего 10. Чтобы избежать подобных ситуаций, некоторые системы обращают внимание на хозяина процесса перед планированием. В такой модели каждому пользователю достается некоторая доля процессора, и планировщик выбирает процесс в соответствии с этим фактом. Если в нашем примере каждому из пользователей было обещано по 50 % процессора, то им достанется по 50 % процессора, независимо от количества процессов.

Планирование в системах реального времени

Известен набор выполняемых задач, поэтому все системы реального времени делятся на 2 категории **гибкие и жесткие** системы реального времени.

В жестких присутствуют жесткие сроки выполнения задачи.

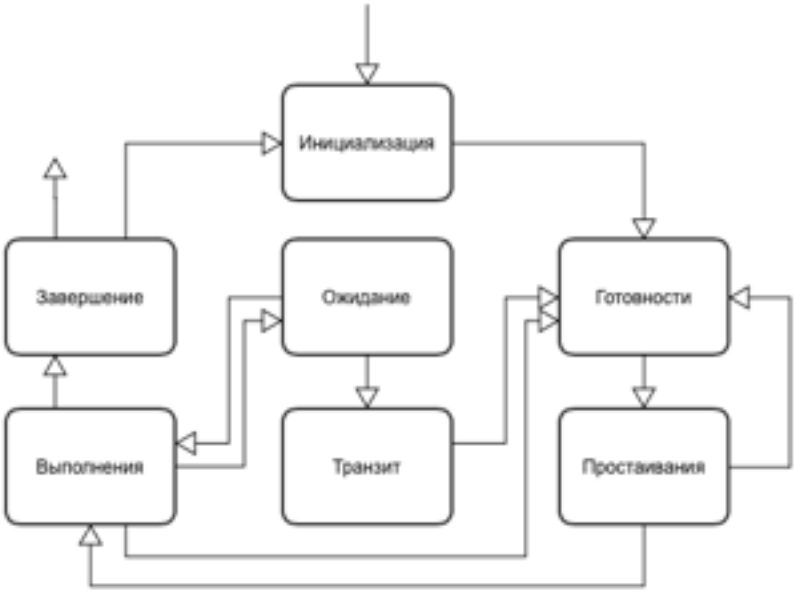
Всем задачам назначаются какие-то статические приоритеты в соответствии с их величиной периода выполнения. Задачи с коротким периодом выполнения получают более высокий приоритет.

В системах реального времени планирование имеет особое значение, поскольку время реакции на сигналы управляемого объекта не должно превышать заданное значение. Планирование облегчается тем, что в системах реального времени весь набор выполняемых задач известен заранее, часто также известно времени выполнения задач, моменты активизации и т. д. Эти данные могут быть использованы для создания статического расписания или для построения адекватного алгоритма динамического планирования. При разработке алгоритмов планирования учитывают, какие последствия возникают при несоблюдении временных ограничений. Если нарушение сроков выполнения задач не допустимо, то система реального времени считается жесткой (система управления ракетой или атомной электростанцией, система обработки цифрового сигнала при воспроизведении оптического диска). Если нарушения временного графика нежелательны, но допустимы, то система реального времени считается мягкой (система продажи билетов).

20 октября

Планирование потока Windows NT

В windows реализован вытесняющий алгоритм планирования на основе уровней приоритета (будет выполняться поток из очереди потоков с наибольшим приоритетом и этот поток будет работать определенный квант времени. При этом, если появится поток с большим приоритетом в очереди готовых, то активный поток вытесняется и выполняется поток с большим приоритетом). Если заканчивается квант времени, то из очереди готовых берется следующий поток с таким же приоритетом, либо будет выполняться этот же поток.



Транзит — готов к выполнению, но стек ядра выгружен, как только загружается переходит в состояние готовности.

Уровень приоритета Windows NT

В системе есть 32 уровня приоритетов. каждый поток может иметь приоритет от 0 до 31.

Приоритеты делятся на 3 группы:

1) 16 - 31 — Приоритеты реального времени
2) 1 - 15 — динамические (варьируемые) уровни
3) 0 — системный уровень

Приоритет каждого потока устанавливается в тот момент, когда происходит создание и инициализация потока, на основе базового приоритета процесса.

Приоритеты:

24 — Реального времени
13 — Высокий приоритет
10 — Выше нормального
8 — Нормальный
6 — Ниже обычного
4 — Простаивающий

У потоков два приоритета базовый (который определяется из приоритета процесса) и текущий (в реальный момент времени, может ±2).

Кванты времени

Квант времени — интервал, промежуток процедурного времени, которое выделяется потоку на выполнение.
В системе Windows разработчики в принципе отошли от использования кванта времени в миллисекундах, придумали систему **Квантовая единица.**
В WinXP квантовая единица == 6
В WinServer квантовая единица == 36

Всякий раз, когда происходит прерывание по таймеру, процедура его обработки вычитает из кванта времени постоянную величину равную **трем.** Величина по таймеру определяется hal.dll

Управление величиной кванта

regedit
HKLM\SYSTEM\CurrentControlSet\Control\ProrityControl\ Win32…
Значение представляет собой 3 битовых поля a1a2a3
 (
	a1 == 0 или 3, то используются кванты времени по умолчанию, т.е. квантовая единица == 6 (XP),
	a1 == 1, то будут использоваться длинные кванты,
	a1 == 2, то кванты будут короткими)
(	a2 == 0 или 3, то используются кванты времени по умолчанию, т.е. кванты динамические (могут меняться), если серверная то они фиксированные
	a2 == 1, то квант активного потока может изменяться
	a2 == 2, то квант активного потока фиксированный)
(	Динамическое приращение квантов
	a3 == 0, 1 или 3, определяют индекс в трехэлементной таблице квантов, используемой для расчета квантов активного потока
	a2 == 3, Недопустимо)

Величина кванта	Короткий	Длинный
Переменный	6 12 18	12 24 36
Фиксированный	18 18 18	36 36 36

Особенности планирования потоков в Windows (разделения времени)

Если после завершения операции ввода/вывода временно повышается приоритет того потока, чтобы быстро возобновить выполнение того потока, который был прерван, но все зависит от того, какое устройство требовалось для ввода/вывода
Порт, диск— прибавляется 1, сеть — 2, клавиатура или мышь — 6, звук — 8.

Диспетчер настройки баланса, который занимается определенным классом задач. Этот диспетчер сканирует очередь готовых к выполнению потоков и если более 300 тактов системного времени поток не выполнялся, то его приоритет становится равным 15 (самый высокий пользовательский приоритет), квант увеличивается в два раза и дается возможность выполниться этому потоку.

Синхронизация процессов и потоков

Проблема синхронизации возникает в многопрограммных ОС.

Чтобы не было гонок и тупиков. Которые обычно возникают при обмене данными между потоками при доступе к процессору и вводу/выводу.

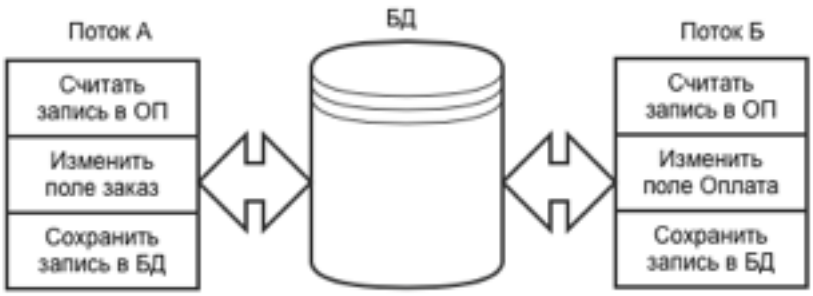
Синхронизация — это приостановка чего-то. Чаще всего синхронизация связана с доступом к каким-то данным.

Понятие гонок

Поток А записывает в БД информацию о поступающих заказах
Поток Б записывает в БД информацию об оплате выставленных счетов
Эти два потока могут работать совместно с БД и по сути они могут совместно обрабатывать одну и ту же информацию.

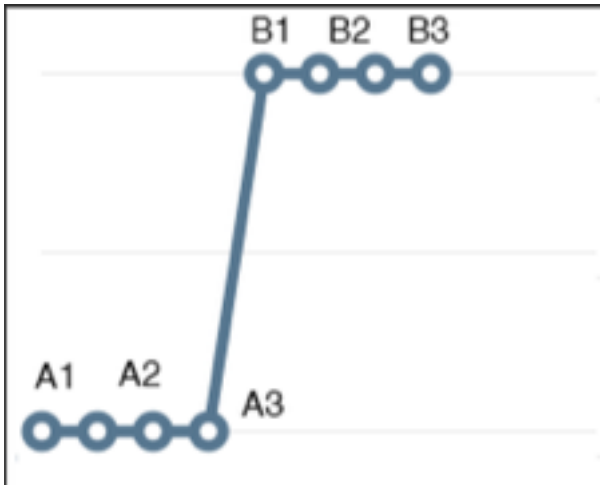
Имеют общий алгоритм:

1) Подключиться к БД
2) Считываем в Оперативную Память из таблицы БД запись о клиенте с его заданным ID
3) Изменяем поля
4) Сохранить в БД



Представим, что клиент под ID == N сделал заказ.

Нормальная ситуация.



Ситуация, когда два или более потока обрабатывают распределяемые данные и конечный результат зависит от соотношения скоростей выполнения потоков **называется гонки**.

Критические секции.

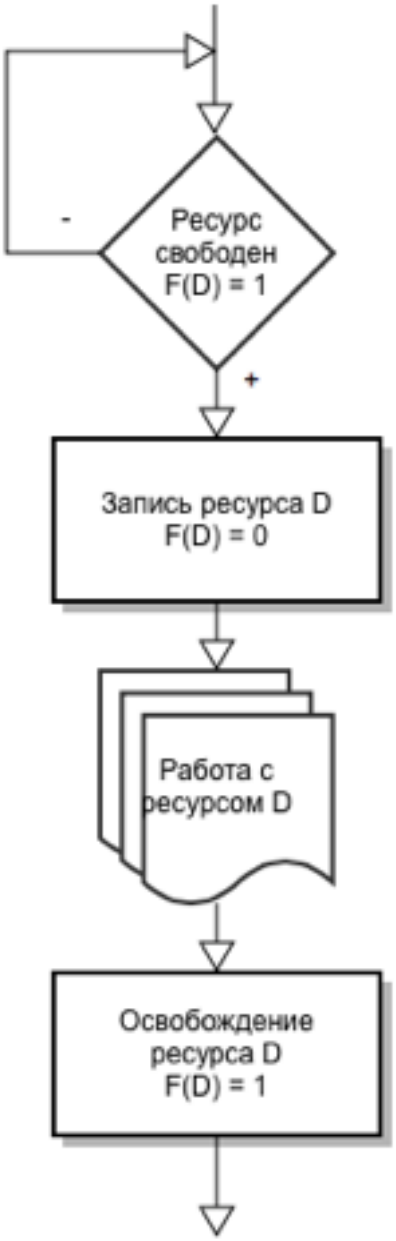
Это часть программы, результат выполнения которого может непредсказуемо изменяться, если переменные относящиеся к этой части программы могут **одновременно изменяться** несколькими потоками.

Весь поток А из предыдущего примера является критической секцией.

В критическую секцию входят критические данные, при несогласованном изменении которых могут возникнуть нежелательные эффекты в виде гонок. Поэтому, во всех потоках, которые работают с критическими данными должны быть определены критические секции. Чтобы исключить эффект гонок по отношению к критическим данным надо сделать так, чтобы с критической секцией мог работать только один поток, и пока этот поток не завершит работать остальные находятся в режиме ожидания.

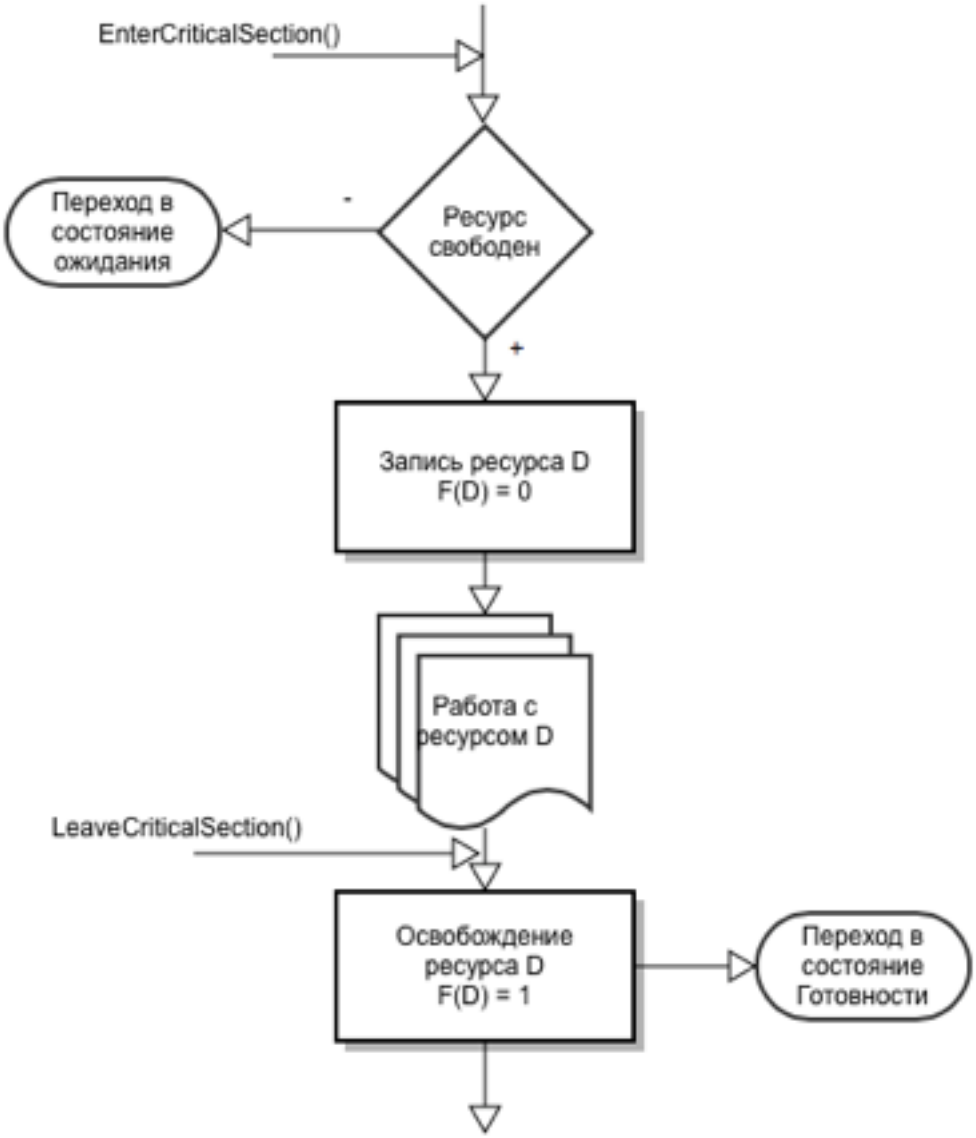
Блокирующие переменные используются для синхронизации потоков только одного процесса. К этим блокирующим переменным должны иметь прямой доступ потоки и этим подходом можно пользоваться без API.

Каждому набору критических данных ставится в соответствие двоичная переменная, которой поток присваивает значение 0, когда он (поток) входит в секцию, и значение 1, когда он ее (критическую секцию) покидает.



Пример реализации критических секций с помощью WinAPI

Желательно пользоваться блокирующими переменными, если функции выполняемые в критической секции малы.



Семафоры

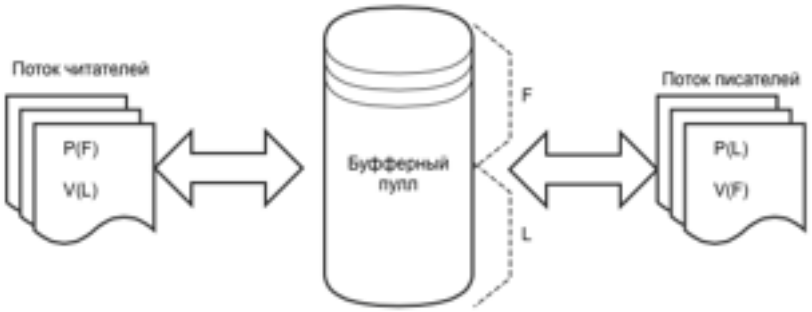
S — семафор
V(S) — Позволяет увеличивать ресурсы семафора на единицу (К этой переменной нет доступа с других потоков)
P(S) — if (S > 0) then S = S - 1 else if s == 0 then (поток переходит в режим ожидания и ждет до тех пор, пока величина S == 0)

Рассмотрим использование семафора на классическом примере взаимодействия двух работающих в режиме мультипрограммирования потоков. Один поток пишет буферный пул, а другой считывает из него. Иногда задачу называют «Писатели и читатели».

L - свободный буфер
F - занятый буфер

Поток писателей сначала выполняет задачу, в которой проверяется имеются ли в пуле свободные буферы, если L == 0, то поток переходит в состояние ожидания, если L > 0, то уменьшает число свободных буферов, записывается информация в очередной свободный буфер и после этого выполняется операция увеличения занятых буферов.

Потоки читателей действуют аналогичным образом, с той лишь разницей, что он вначале начинает проверку наличия заполненных буферов, если F == 0 — ожидаем , иначе читаем и увеличиваем количество свободных.



Т.к. семафоры позволяют организовать доступ к критическому ресурсу более, чем одному потоку.

Взаимоблокировки(deadlock)

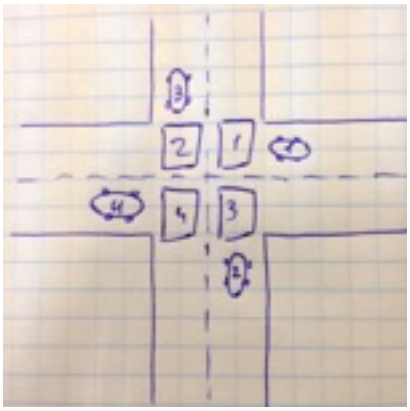
10 ноября

Взаимоблокировку можно определить, как постоянное блокирование множества процессов, которые конкурируют в доступе к системным ресурсам компьютера, либо при взаимодействии с друг другом.

Классический пример: транспортная блокировка.

1) 1, 2
2) 3, 4
3) 2, 1
4) 4, 3

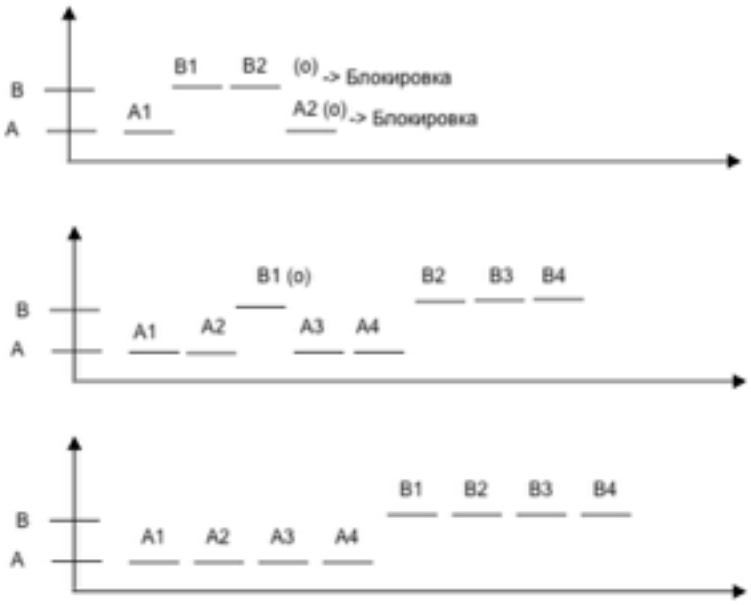
Ситуация предрасположена к блокировке



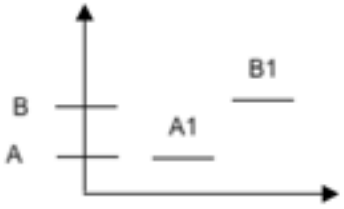
Блокировка возникнет, когда машины займу позиции:

1) 1
2) 3
3) 2
4) 4

Рассмотрим ситуацию близкую к блокировке. Есть два процесса и им необходимо выполнить одну работу вывод на принтер и порт



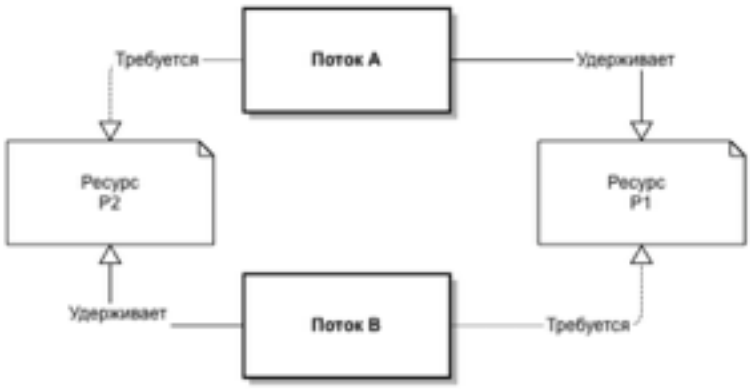
Когда у нас есть два одинаковых процесса и каждый из этих процессов записывает отсканированный документ на какой-то диск.



Выгружаемый ресурс, это ресурс, который может быть безболезненно забран у процесса и переден другому процессу.
Не выгружаемый ресурс — невозможно забрать у текущего владельца не забрав результат каких-то действий.

Хоффман написал условия возникновения ситуаций взаимоблокировки в ОС:
1) **Условие взаимного исключения** (Ресурс либо отдан одному какому-то потоку, либо свободен)

2) **Условие удержания и ожидания** (Любой процесс, который удерживает какие-то ресурсы может запросить новый, необходимый ему ресурс)
3) **Условия отсутствия принудительной выгрузки ресурсов** (У процесса нельзя принудительно забрать ресурс, а процесс освобождает этот ресурс в ходе выполнения своих действий)
4) **Условие циклического ожидания** (Должна существовать круговая последовательность из двух или более потоков, каждый из которых ждет доступа к ресурсу удерживаемому следующим потоком в этой последовательности)
   Должна существовать замкнутая цепь потоков, каждая из которых удерживает один ресурс, необходимый потоку следующему в этой цепи после данного потока.



Если количество ошибок, которые возникают в случае сбоев оборудования соизмеримо с количеством взаимоблокировок, то используется **страусовый алгоритм.**

Выход из взаимной блокировки

Если блокировка обнаружилась, то требуется некая стратегия для возврата ОС в нормальный режим:

1) Прекратить и перезапустить все процессы
2) Сделать откат на какой-то интервал времени заблокированных процессов и разрешить им выполняться (Должен быть механизм транзакций для возврата и нет гарантий, что не попадем опять в блокировку)
3) Последовательно прекращать работу заблокированных процессов по одному, до тех пор, пока не исчезнет ситуация блокировки
4) Снимать процесс, с минимальным выводом информации или с наибольшим временем ожидания, или с минимальным количеством захваченных ресурсов, или с минимальным приоритетом
5) Последовательно распределять ресурсы, пока взаимоблокировка не прекратится

Алгоритм Банкира

A	0	6
B	0	5
C	0	4
D	0	7

Алгоритм Банкира рассматривает каждый запрос по мере его поступления и проверяет, приведет ли его удовлетворение к безопасному состоянию, если да, то процесс получает ресурс, иначе запрос переносится на более позднее время.

Управление памятью

Память — важный ресурс, который требует тщательного управления.

Базовые задачи:

1) Отслеживает свободную и занятую память
2) Выделяет память процессам и освобождает память после освобождения процесса
3) Вытесняет коды и данные процессов из оперативной памяти, когда размер ОП недостаточен для размещения в ней всех процессов и возвращает их обратно ОП, когда в этом возникает необходимость
4) Настраивает адреса программ на конкретную область физической памяти

Типы адресов

1) Символьные адреса (Те имена, которые присваивает пользователь при написании программы)
2) Виртуальные адреса (Их вырабатывает транслятор, переводящий программу в машинный код. Транслятор присваивает переменным и командам виртуальные адреса, при этом обычно начальный адрес программы нулевой)
3) Физические адреса (Соответствуют номерам ячеек оперативной памяти, где в действительности расположены переменные команды в ходе выполнения команд препроцессора)

Совокупность виртуальных адресов процесса называется виртуальным адресным пространством (ВАП). Максимальный диапазон виртуального адресного пространства всех процессов от 0 до 2^32

При сегментной памяти ВАП делится на части и любой адрес содержит номер сегмента и смещение 0000:FFFF

При плоской памяти адресом является единственное число.

Существует два подхода преобразования виртуального адреса в физический:

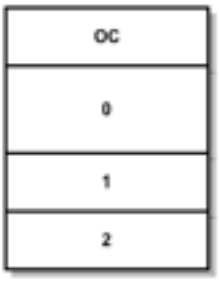
1) Замена виртуального адреса на физический выполняется один раз для каждого процесса в момент начальной загрузки процесса в ОП
2) Программа загружается в ОП в не измененном виде в виртуальных адресах. Но при загрузке этой программы в ОП фиксируется начальный адрес загрузки этой программы. Во время выполнения программы при каждом обращении к ОП выполняется преобразование виртуального адреса в физический.

17 ноября

Методы распределения памяти

Алгоритмы без использования виртуальной памяти

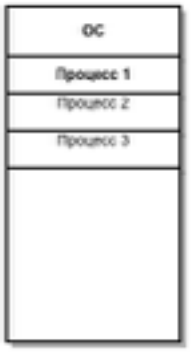
Распределение с фиксированными разделами:



* Память выделяется под ОС, а оставшаяся память разделяется на несколько областей фиксированной величины, границы разделов не изменяются

Подсистема выделения памяти работает следующим образом: Для очередной задачи, которая поступила на выполнение предварительно будет выбран раздел, в котором может быть размещена эта задача.
Загрузка программы в выбранный раздел и осуществляется настройка адресов.
Достоинства: Простота
Недостатки:Не используется вся область памяти; в разделе может выполняться одна задача

Распределение памяти с динамическим разделом:
Каждому вновь поступившему на выполнение процессу, выделяется необходимая оперативная память



Подсистема управления:
1) Должна вести таблицы свободных и занятых областей, в которых указываются начальные границы и размеры участков памяти
2) При создании нового процесса осуществляется анализ требований к памяти, рассматривается таблица свободных областей и выбирается раздел, размеры которого достаточны для размещения кода и данных нового процесса.
3) Выполняется загрузка программы в выделенный ей раздел и выполняется корректировка таблиц свободных и занятых областей
4) Для определения физического адреса (af) мы должны знать адрес загрузки нашей программы и ее смещение данного процесса (af = as + S)

Фрагментация — наличие большого количества маленьких участков свободной памяти

Распределение памяти с использованием перемещаемых разделов:
Дефрагментация — перемещение всех разделов в область максимальных или минимальных разрядов (Приостанавливаем все процессы и переносим)

Алгоритмы с использованием виртуальной памяти (внешней памяти)

Свопинг — это когда целиком весь выполняемый процесс помещается на диск, а когда понадобился считывается обратно
Виртуализация — когда между ОП и диском перемещаются части образов процессов

Из всего множества алгоритмов работы в ВП можно выделить 3 различных класса виртуализации:

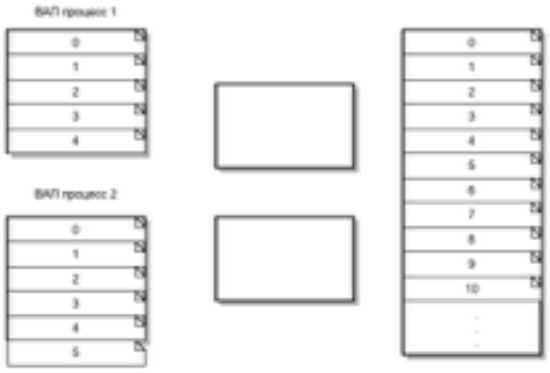
1) Страничное распределение
   Когда перемещение данных между ОП и дисками выполняется страницами, т.е. частями виртуального адресного пространства фиксированного и сравнительно небольшого размера
2) Сегментная виртуальная память, которая предусматривает перемещение сегментами, т.е. частями виртуального адресного пространства произвольного размера полученным с учетом полученных смысловых данных (Компилятор делит на сегменты)
3) Сегментно-страничная виртуальная память, которая использует двух-уровневое деление
   1) Виртуальное адресное пространство делится на сегменты
   2) Сегменты делятся на страницы

Страничное распределение памяти (алгоритм)
Виртуальное пространство каждого процесса делится на части одинакового, фиксированного для данной ОС размера, называемого виртуальными страницами

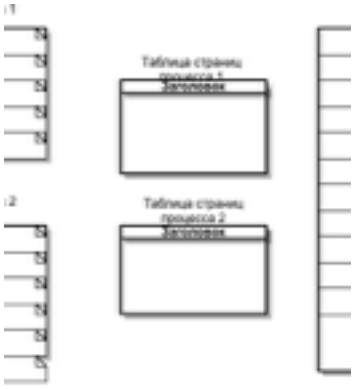
Вся ОП делится на части такого же размера и эти части буду называться страницами (физические страницы)

Виртуальные странички будут грузиться в физические

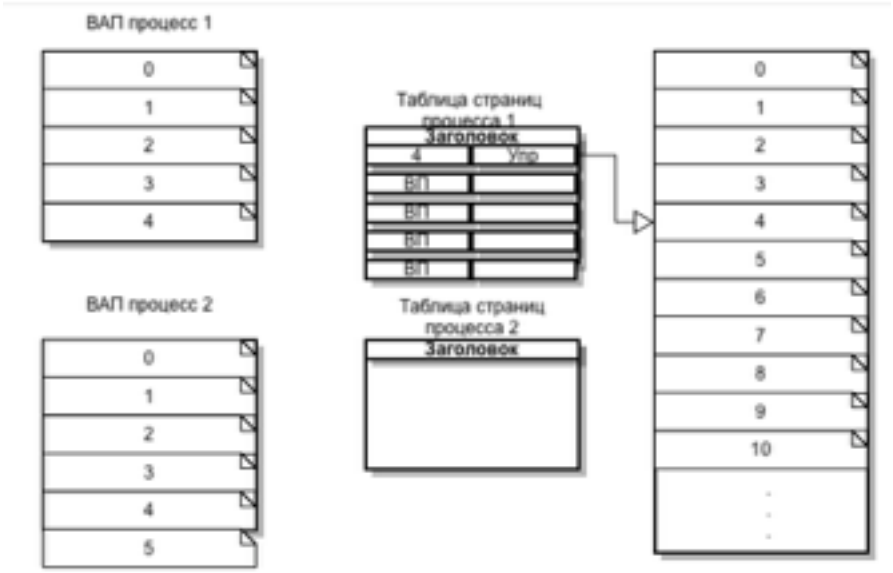
Важна скорость, поэтому выбирается размер страничек 2^k



Для каждого процесса ОС создает таблицу страниц, это информационная структура содержащая записи, обо всех виртуальных страницах процесса



Запись этой таблицы называется дескриптором таблицы и она включает в себя:
1) Номер физической страницы, в которую загружена данная виртуальная страница
2) Управляющая информация
    1) Признак присутствия (1 — виртуальная страница размещена в ОП)
    2) Признак модификации (равняется 1 каждый раз, когда по адресу относящемуся к этой странице происходит модификация данных)
    3) Бит доступа (1 — каждый раз, при обращении к адресу относящемуся к этой странице)



Информация из таблицы страниц используется для решения вопроса о необходимости перемещения той или иной страницы из ОП на диск, а так же для преобразования виртуального адреса в физический

Предположим, что происходит обращение по какому-то адресу, вначале анализируется признак присутствия.
Если страница находится в ОП, то сразу реализуется алгоритм преобразования виртуального адреса в физический (рассмотрим ниже).
Если страница с данным виртуальным адресом выгружена на диск, то происходит так называемое страничное прерывание.

Страничное прерывание запускает процесс по нахождению необходимой страницы на диске и загрузке ее в ОП. Процесс переходит в состояние ожидания, пока механизм, которым управляет ОС не закончится.

Если в ОП есть свободные физические страницы, то загрузка с диска выполняется немедленно, а затем реализуется алгоритм преобразования виртуального адреса в физический.

Если же **необходимой памяти нет**, то ОС решает вопрос о том, какую (какие) страницы выгрузить из ОП на диск. Найдя страницу …

Если бит модификации равен единице, то ее новая версия должна быть переписана с ОП на диск

Виртуальный адрес представляет собой номер виртуальной страницы и смещение в этой виртуальной страничке.

Основные свойства страничной организации:

1) Смещение (S) внутри каждой страницы определяется из текущего адреса, из k разрядов в двоичном представлении
2) Из любого виртуального адреса, чтобы определить номер страницы надо сдвинуть этот адрес в право на k разрядов
3) Чтобы определить начальный адрес виртуальной страницы (базовый адрес страницы), нужно k разрядов справа заменить нулем
4) В пределах страниц присутствует непрерывная последовательность виртуальных адресов, которая однозначно отображается в непрерывную последовательность адресов физической страницы (адрес виртуальный≠адресу физическому, а смещения равны)

Сегментная организация:

Все множество виртуального адресного пространства делится на разделы (каталоги), а разделы потом делятся на страницы.

m -- количество позиций для номера раздела, на которые разобьется ВАП.
k -- количество позиций для номера страницы в данном разделе.
n -- разрядность (смещение) каждой страницы.

Номер последнего раздела 2^m - 1
Номер последней странички раздела 2^k - 1.

Все страницы имеют одинаковый размер, а все разделы содержат одинаковое количество страниц.

Размер страницы -- 2^n
Количество страниц в разделе -- 2^k
Количество разделов -- 2^m

Виртуальный адрес можно будет представить следующим образом: n разрядов будут представлять смещение виртуальной страницы, k -- номер страницы в разделе, m -- номер раздела.

Для каждого раздела строится собственная таблица страниц, и количество дескрипторов в этой таблице и их размер подбираются. таким образом, чтобы объем этой таблицы был равным объему виртуальной страницы.

На адресацию внутри страницы отводится 4 Кб, значит n = 2^12
k = 2^10
m = 2^10

Рассмотрим схему преобразования адресов для случая двухуровневой структуризации виртуального адресного пространства.

Виртуальный адрес состоит из трех частей:
1. Номер раздела.
2. Номер страницы.
3. Смещение.

Физический адрес состоит из двух частей:
1. Номер страницы.
2. Смещение.

Предположим, что происходит обращение по какому-то адресу:
1. Определяется номер раздела, к которому принадлежит данный виртуальный адрес путем отбрасывания k+n младших разрядов этого адреса.
2. Из таблицы разделов извлекается дескриптор соответствующей таблицы страниц.

Проверяем, находится ли данная таблица страниц в оперативной памяти. Если нет, то происходит страничное прерывание, данная таблица находится (в смысле ищется и находится:)) в файле подкачки и загружается в ОП.

3. Из данной таблицы извлекается дескриптор страницы и совершается переход по адресу нахождения этой страницы.
4. Из данной таблицы страниц извлекается дескриптор из которого извлекается номер физической страницы.

Оптимальный алгоритм замещения страниц
Предполагается, что в памяти находится какой-то набор страниц. К каждой из этих страниц будет обращаться в какой-то момент времени какой-то процесс. Каждая страница может быть помечена числом равным количеству команд, которые будут выполняться перед первым обращением к этой странице.

Оптимальный алгоритм будет удалять страницы с максимальным числом.

Реально используемые алгоритмы:
NRU (Not recently used) -- исходная информация -- таблица страниц.

Выделяют два бита: R -- обращение, M -- изменение.

Когда какой-то процесс запускается, R == M == 0. В дальнейшем периодически (при каждом, например, прерывании по таймеру) бит R очищается для того, чтобы можно было отличить страницы к которым давно не происходило обращение от тех, на которые были ссылки.

Когда происходит страничное прерывание ОС проверяет дескрипторы всех страниц и делит их на четыре категории (классы) в зависимости от значений дескрипторов R и M.

Класс 0 -- не было обращений и изменений. 00.
Класс 1 -- не было обращений -- но страницы изменены. 01.
Класс 2 -- было обращение, но страница не изменена. 10.
Класс 3 -- изменение и обращение. 11.

1 декабря

Алгоритмы замещения страниц

FIFO

Новая страница загружается в хвост списка, а из головы выгружаем.

Вторая попытка

Модификация алгоритма FIFO, в том, что алгоритм FIFO может выгрузить из памяти часто загружаемые страницы.
Используется бит R (бит обращения).

Если R == 0, то к этой странице давно не было обращений, поэтому ее можно заменить или убрать, если же R == 1, то ему присваивается значение 0, а страница переносится в конец списка.

Часы

Алгоритм «**Вторая попытка**» является не совсем эффективным потому, что в нем постоянно перемещаются страницы по списку. Предлагается хранить все страничные блоки в кольцевом списке в форме часов. Стрелка в часах указывает на старейшую страницу.
Если происходит страничное прерывание, то проверяется страница на которую указана стрелка, если ее бит R == 0, то страница удаляется, иначе он приравнивается 0 и стрелка перемещается. Таким образом стрелка дойдет до той, у которой R == 0 и она заменяется.

LRU

Least Recently Used (Страница не использовавшаяся дольше всего)
Предположим, что наше ВАП из 4 страниц, и следующий порядок обращения к страницам:
0 1 2 3 2 1 0 3 2 3
Создается матрица NxN бит и первоначально все элементы == 0

И когда происходит обращение к i -ой странице, то все биты строки i присваиваются 1, а все биты столбца 0

После обращения к 0

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

После обращения к 1

	0	1	2	3
0	0	0	1	1
1	1	0	1	1
2	0	0	0	0
3	0	0	0	0

После обращения ко 2

	0	1	2	3
0	0	0	0	1
1	1	0	0	1
2	1	1	0	1
3	0	0	0	0

Если возникает страничное прерывание, то удаляем страницу с наименьшим значением в строке.

NFU

Not Frequently Used (Редко используемая страница)

Для каждой страницы находящейся в ОП определяется счетчик, который вначале равен нулю. При каждом прерывании по таймеру проверяется значение бита R каждой страницы и его значение прибавляется к счетчику. Если бит R ==0, то ничего к счетчику не прибавляется.

При страничном прерывании для замещения выбирается страница с наименьшим значением счетчика.

Алгоритм «Старения»

Является доработкой алгоритма **NFU** и суть этого алгоритма заключается в следующем:

1) Каждый счетчик сдвигается вправо на 1 разряд перед прибавлением бита R
2) Бит R прибавляется в крайний слева, а не справа разряд этого счетчика

Когда происходит страничное прерывание — удаляется та страница, которая имеет наименьшее бита R

Аномалия Билэди

Возникает вопрос, «**Сколько должно быть в ОП физических страниц, чтобы система функционировала эффективнее?**»

Рассмотрим пример.

Виртуальное адресное пространство нашего процесса будет сдержать 5 страниц.

Обращение будет происходить в следующем порядке: 0 1 2 3 0 1 4 0 1 2 3 4

Предположим, что у нас может быть 3 или 4 физических страницы. Алгоритм замещения страниц **FIFO**

Номер физ страницы	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	0	1	4	4	4	2	3	3
1		0	1	2	3	0	1	1	1	4	2	2
2			0	1	2	3	0	0	0	1	4	4
	прерывание	прерывание	прерывание	прерывание	прерывание	прерывание	прерывание			прерывание	прерывание	

9 прерываний

Предположим, что теперь у нас может быть 4 физ страницы

Номер физ страницы	0	1	2	3	0	1	4	0	1	2	3	4
0	0	1	2	3	3	3	4	0	1	2	3	4
1		0	1	2	2	2	3	4	0	1	2	3
2			0	1	1	1	2	3	4	0	1	2
3				0	0	0	1	2	3	4	0	1
	прерывание	прерывание	прерывание	прерывание			прерывание	прерывание	прерывание	прерывание	прерывание	прерывание

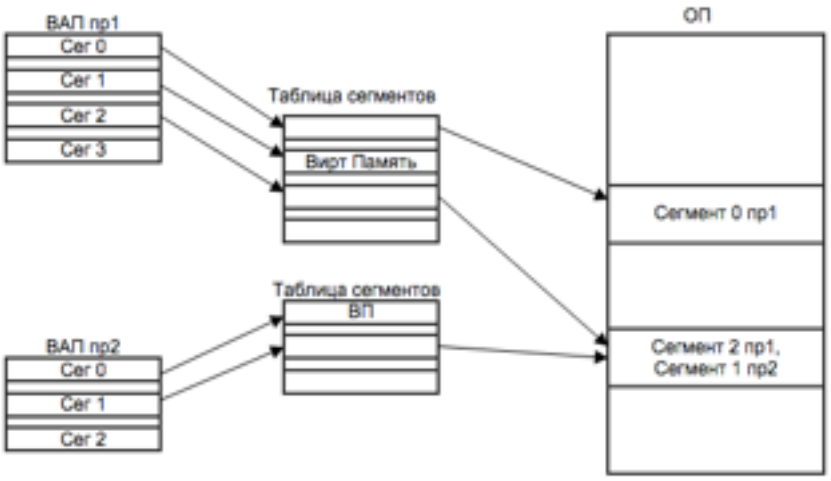
10 прерываний

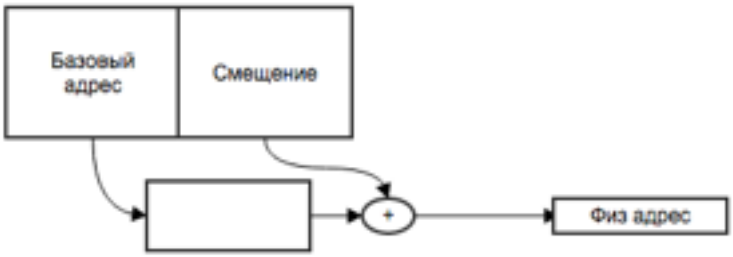
Сегментное распределение памяти

ВАП каждого процесса разбивается на сегменты, размеры которых определяются с учетом смыслового значения содержащихся в них информации.
Сегмент может представлять собой подпрограмму, функцию, массив. Деление виртуального пространства на сегменты определяет компилятор. Максимальный размер сегмента в 32 разрядной системе может быть 4 Гб.

Виртуальный адрес = номер сегмента + линейный виртуальный адрес внутри этого сегмента. (V->n, S)
При загрузке процесса в ОП загружается только часть сегментов, а все ВАП загружается в файл подкачки. Если во время выполнения процесса происходит обращение к другому сегменту, который в данный момент отсутствует в ОП, то данный процесс переводится в состояние ожидания, возникает прерывание и ОС загружает необходимый сегмент с диска в ОП, если необходимой памяти в ОП нет, то ОС так же должна решить вопрос о выгрузке какого-то сегмента из ОП на диск.

В момент создания процесса, в ОП так же создается таблица сегментов.
В дескрипторах этой таблицы указывается следующее:
1) Базовый физический адрес сегмента в ОП (Если сегмент размещен в ОП)
2) Размер сегмента
3) Права доступа к сегменту
4) Управляющая информация (модификация, присутствие, обращение)





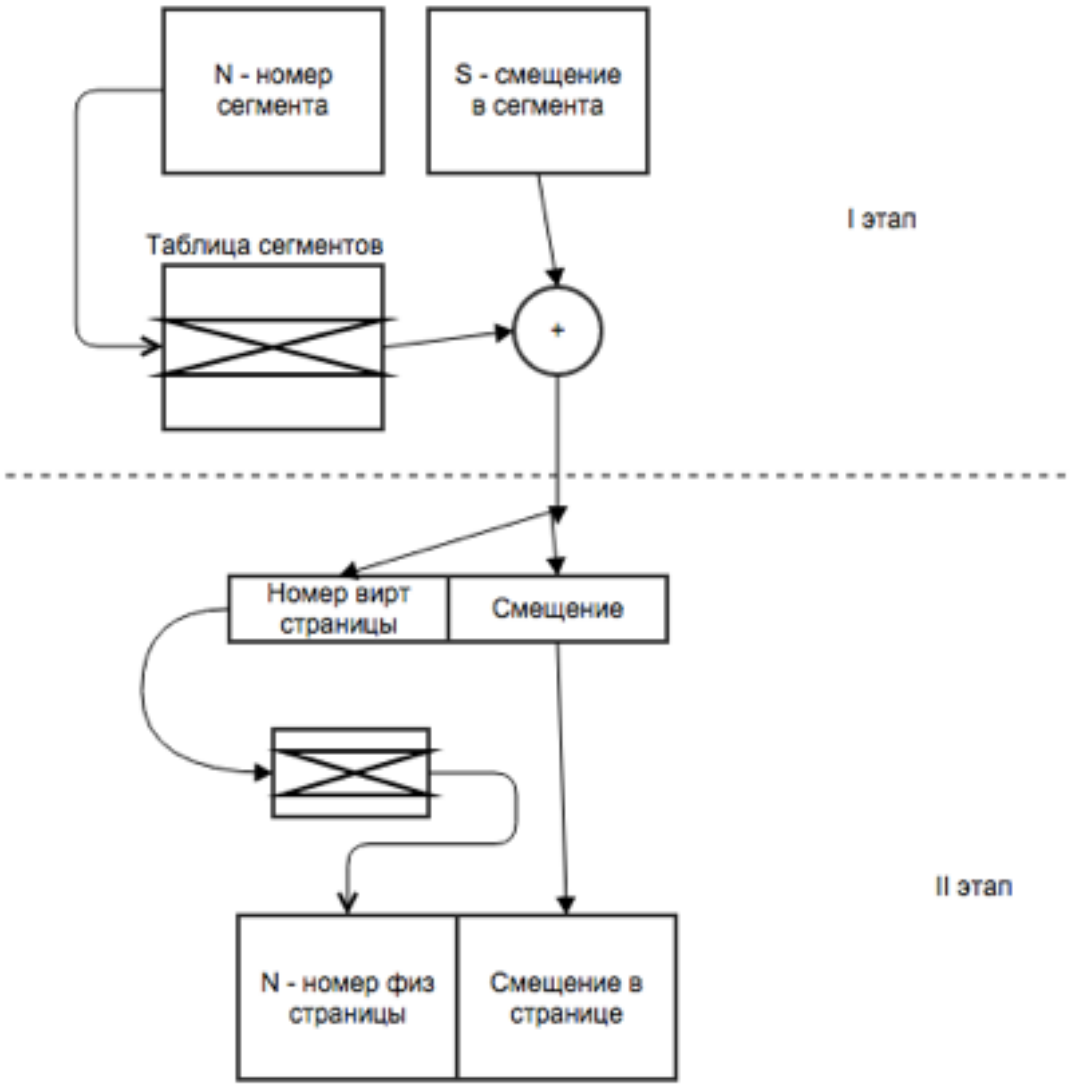
Сегментно-страничное распределение памяти

Сегментно-страничное распределение является комбинацией страничного и сегментного методов управления памятью.
Сначала ВАП[1] делится на сегменты, а затем сегменты делятся на страницы равного размера. Перемещение между ОП и диском осуществляется страницами.

Для каждого ВАП создается таблица сегментов (дескрипторов = кол-ву сегментов), однако в поле базового адреса указывается не начальный физический адрес этого сегмента в ОП, а начальный линейный виртуальный адрес сегмента в пространстве виртуальных адресов. Наличие базового виртуального адреса сегмента позволяет однозначно преобразовать адрес заданный в виде пары чисел (Сегмент:Смещение) в линейный виртуальный адрес, который затем преобразуется в физический адрес механизмами страничного распределения. Размер страницы выбирается равным степени двойки, что значительно упрощает механизм преобразования в **Виртуальный линейный адрес**.
При создании процесса в ОП загружается только часть страниц, а остальные страницы подгружаются по необходимости, согласно алгоритмам замещения страниц. Базовые адреса сегментов и таблицы страниц процесса являются частью его контекста.
При активизации процесса эти адреса загружаются в специальные регистры процессора, которые затем будут использоваться для преобразования адреса виртуального в физический.

На первом этапе работает механизм сегментации. Исходный виртуальный адрес заданный N и S преобразуется в линейный виртуальный адрес, для этого на основании базового адреса в таблице сегментов и номера сегмента вычисляется адрес дескриптора.
На втором этапе работает страничный механизм. Номер вирт страницы — старшие биты 2^k

[1] Виртуально адресное пространство



Файловая система

Требования к файловой системе

1) Возможность хранить большие объемы данных
2) Информация должна сохраняться после прекращения работы
3) Несколько программ могут получить доступ

Цели использования файлов

1) Долговременное и надежное хранение файлов
2) Совместное использование информации

Файловая система — часть ОС, которая включает в себя:

1) Совокупность всех файлов на диске
2) Наборы структур данных используемых для управления файлами (дескрипторы файлов, каталоги файлов, таблица занятого и свободного места в памяти)
3) Комплекс системных программных средств, реализующих действия над файлами (создание, уничтожение, чтение, запись, переименование, поиск файлов)

Типы файлов:

1) Регулярные (обычные)
2) Каталоги

Регулярные файлы — содержат информацию произвольного характера, которую заносит в них пользователь или которая появляется в них в результате работы системных или пользовательских приложений.

Каталог — особый вид файла, который содержит системную справочную информацию о наборе файлов. Устанавливают связь между именами файлов и их характеристиками, используемых файловой системой для управления этими файлами.

Теоретическая структура файловой системы

Дерево —файл входит только в один каталог
Сеть — файл может находиться в нескольких каталогах

Выделяют три типа имен файла

1) Простое (идентифицирует файл в пределах того каталога, где он находится. Вид определяет ОС)
   MS DOS 8 латинских символов под имя, 3 под расширение
   UNIX 14 латинских символов под имя
   NTFS, FAT32 256 символов под имя
2) Составное (Цепочка имен всех каталогов)
3) Относительное (Через понятие к текущему каталогу)

Каталог в MS DOS 32 символа

имя файла	расширение	R	A	H	S	резерв	время	дата	N первого кластера	Размер файла
8	3	1				10	2	2	2	4

Каталог в UNIX 16 символов

N дескриптора	Имя файла
2	14

Атрибуты файла — информация, описывающая свойства файлов

В NTFS есть права доступа, в FAT их нет

Логическая организация файлов

Данные, которые находятся в файлах могут иметь определенную логическую структуру. Поддержание структуры данных в файлах может быть возложена либо на приложения, либо на ОС.

Физическая организация файловых систем

Жесткий диск состоит из нескольких пластин

1) Номер цилиндра (Цилиндр — совокупность дорожек с одинаковыми номерами на всех пластинах. Каждая дорожка делится на сектора)
2) Номер поверхности
3) Номер сектора (Наименьшая единица объема при записи информации на диск. Размер 2^9 = 512 байт)

ОС при работе с диском использует свою собственную единицу дискового пространства, которая называется **кластер.** Кластер состоит из нескольких секторов. Размер кластера из 2^k байт (минимум k = 9)

Два типа форматирования:

1) Физическое (Производителем 1 Тб, 2 Тб и т.д.)
2) Логическое (разметка диска на логические разделы, определяется размер кластера)

MDR формируется во время логического форматирования, один из разделов должен быть определен как активный.

Основными критериями эффективности физической памяти является:

1) Скорость доступа к данным
2) Объем адресной информации файла
3) Степень фрагментированности адресного пространства
4) Максимально возможный размер файла

Способы физической организации:

1) Непрерывное размещение (выбираем последовательные кластера)
2) Размещение файла в виде связанного списка кластеров дисковой памяти (вначале каждого кластера разместить указатель на следующий кластер. Сложность обращения к любой части файла. Количество файлов данных хранящихся в кластере не кратно степени двойки)
3) Использование связанного списка индексов (Модификация предыдущего способа. Файлу так же выделяется память в виде связанного списка кластеров, номер первого кластера запоминается в записи каталога, где хранятся все атрибуты этого файла, а номера следующих кластеров этого файла хранятся в специальной таблице индексов. В этой таблице индексов с каждым индексом связан определенный кластер. Индексы располагаются в отдельной области диска и когда кластер свободен значение индекса равно 0. Если некоторый кластер назначается какому-то файлу, то индекс этого кластера становится равным номеру следующего кластера этого файла, либо принимает специальное значение указывающее, что этот файл закончился)

15 декабря

4) Перечисление номеров кластеров
   Если записали информацию в 1, 4, 5 кластера, то адрес файла будет перечислением кластеров 1,4,5
   Быстрый доступ, нет фрагментации, большой объем адресной информации (заранее размер не известен)

   Рассмотрим модификацию, используемую в системах UNIX.
   Для сохранения объема адресной информации можно сочетать с косвенной адресацией. Для хранения адреса файла выделяется 15 полей (60 байт)

0	1	2	…	10	11	12	13	14

   Если размер файла меньше или равен 12 кластерам, то номера этих кластеров непосредственно перечисляются в первых 12 полях. В системе UFS предполагалось, что размер кластера равен 8 Кб = 8192 байта
   Если размер файла превышает 12 кластеров, то используется 13 поле, которое содержит адрес кластера, в котором могут располагаться следующие адреса кластеров файлов (2048 номеров кластеров, которые указывают на продолжение файла). 13 элемент адреса используется для косвенной адресации.

   Всего кластеров может быть (12 + 2048) * 8192 = 16 875 520 байт

   Если размер файла превышает 2060 кластеров, то используется 14 поле для хранения 2048 адресных кластеров. (12 + 2048 + 2048 * 2048) * 8192 = 4 196 364 * 8192 = 33 570 912 Кбайт

   15 элемент используется тройная косвенная информация

   У данного метода была дальнейшая модификация (являлась прототипом для NTFS). В модификации сокращается объем информации, за счет того, что адресуются не

номера кластеров, а непрерывные области, состоящие из смежных кластеров диска (вместо номера кластера используется два числа, номер кластера и количество номеров в этой области).

Файловая система FAT

MS DOS FAT

Логический раздел делился на 5 областей:

1) Загрузочный сектор 512 байт
2) Таблица FAT (содержит информацию о размещении файлов и каталогов на диске) Состоит из массива индексов указателей, количество которых равно количеству кластеров области данных. Между кластерами и индексными указателями имеется взаимно однозначное соответствие. Индексный указатель может принимать следующие значения:
    1) кластер свободен
    2) кластер используется файлом и не является последним кластером файла, то в этом случае индекс содержит номер следующего кластера файла
    3) Последний кластер файла
    4) Дефектный кластер
    5) Резервный кластер
3) Резервная копия FAT
4) Корневой каталог (32 сектора, 16 Кб)
5) Область данных (делится на кластера)

При создании файла происходит просмотр таблицы, если находит свободный, то файловая система фиксирует этот номер в каталоге (4 пункт), и записывает номер первого кластера для файла. В области данных в 16 кластер заносит информацию.

FAT-12, FAT-16, FAT-32 — максимальное количество кластеров 2^

Физическая организация UFS

На диске 4 области. Диск — том, кластер — блок.

1) Загрузочный блок
2) Супер блок, который содержит информацию о файловой системе (размер, размер области, индексный дескриптор, число дескрипторов, список свободных блоков, список свободных индексных дескрипторов)
3) Область индексных дескрипторов (Дескрипторы размещаются по возрастанию)
4) Область файлов (обычные файлы и файлы каталоги)

Имя файла отдельно от его характеристик.

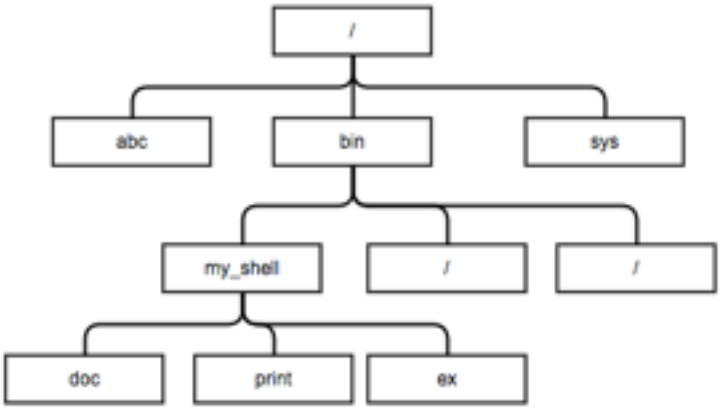
Каталог в виде таблицы, где 14 байт отводилось под имя и 2 под номер дескриптора.

Размер одной записи 64 байта (идентификатор владельца файла, тип файла, права доступа к файлу, временные характеристики (модификация, обращение, …), адресная информация, размер файла).

Каждый индексный дескриптор имеет уникальный номер, который по сути является уникальным именем файла. Соответствие, между именем и номером определяется при помощи иерархии каталогов. При сохранении файла ему выделяется из списка номер свободного индексного дескриптора, а при уничтожении этого файла этот номер индексного дескриптора возвращается в список.

Пример.

Обращение к /bin/my_shell/print



1) Просматривает корневой каталог с целью определения первой составляющей и составляется индексный дескриптор каталога
2) Из области индексных дескрипторов считывается дескриптор с номером 6.

abc bin sys	2 6 11

Из индексного дескриптора номер 6 определяем физический адрес каталога 2

3) Просматривает новый каталог с целью определения следующей составляющей и составляется индексный дескриптор каталога
4) Из области индексных дескрипторов считывается дескриптор с номером 25.

my_shell … …	25 35 36

5) Просматривается каталог /bin/my_shell/ и определяется файловый дескриптор файла

doc print ex	176 131 172

6) Из индексного дескриптора с номером 131 определяем номер блока, в котором находится информация файла print

NTFS

До нее была HPFS (Hight Perfomance File System).
NTFS — New Technology File System.

Требования:
1) Больших файлов и дисков, объемом до 2^64
2) Восстановление после сбоев, отказов программных и аппаратных средств
3) Высокая скорость операций
4) Низкий уровень фрагментации
5) Гибкая структура, допускающая добавление новых атрибутов
6) Поддержка длинных символьных имен
7) Поддержка доступа к каталогам и отдельным файлам
8) Возможность автоматического сжатия файлов
9) Возможность шифрования файлов
10) Квотирование пользователей (Какой объем разрешить пользователю)

Физическая организация

Логический диск разбивается на:
1) Загрузочный сектор
2) Главная файловая таблица (MFT)
3) Системные файлы
4) Область данных, в которой повторяется Загрузочный сектор

MFT является таблицей, строки в которой являются записями с номерами. Все номера на диске идентифицируются с дескриптором, для которого используется 64 разряда и его номер определяется в таблице MFT.

Первый отрезок MFT содержит 16 стандартных записей, созданных при форматировании

Отрезок — несколько смежных кластеров. Есть первый номер и количество кластеров, которые входят в отрезок.

Адресация происходит по 3 числам
VLN (Виртуальный текущий номер, смещение относительно начала), LCN (логический номер кластера), K (количество кластеров)

Запись
1) если файл маленький то записывается прямо в MFT (~1.5 Кб)
2) Большой файл в области данных MFT (Пишутся адреса данных)
3) Файл очень большой, то в области данных MFT адрес на блок с адресами данных
4) Огромный файл, , то в области данных MFT адреса на блоки с адресами данных