1. Изобразить структуру операционной системы. Объяснить назначение компонентов.

2. Изобразить систему очередей планирования и объяснить принцип работы.

3. Приведите основные стратегии планировщиков.

4. Описать процесс свопинга. Перечислить достоинства и недостатки.

5. Сформулируйте основные функции операционных систем.

6. Приведите классификацию ресурсов вычислительной системы.

7. Опишите эволюцию структур операционных систем. Приведите структурные схемы.

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

9. Изобразите схему преобразования адреса для платформы Intel. Опишите элемент PTE.

10. Перечислите типы операционных систем и их основные отличительные свойства.

11. Постройте схему структурирования фиксированными страницами и опишите процесс преобразования адресов.

12. Сформулируйте задачи и функции файловой системы.

13. Изобразите схему структурирования переменными страницами и опишите процесс преобразования адресов.

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

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

16. Сформулируйте задачи управления виртуальной памятью.

17. Опишите виды групп информации дескриптора.

18. Приведите характеристики файлов и типы доступа к файлу.

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

20. Приведите классификацию процессов.

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

22. Приведите структуры организация файла логического уровня.

23. Вытесняющие алгоритмы планирования процессов.

24. Права доступа к файлу. Основные подходы к определению прав доступа.

25. Механизм кэширования диска. Достоинства и недостатки.

26. Приведите результаты сравнительного анализа NTFS и ReFS.

27. Задачи и свойства распределенных файловых систем.

28. Опишите архитектуру современной файловой системы.

29. Опишите механизмы синхронизации и взаимодействия процессов.

30. События. Семафоры. Мьютексы. Сообщения. Их основное назначение.

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

32. Провести сравнительный анализ ОС Linux и Windows по функционалу систем.

33.Провести сравнительный анализ Android и Windowsпо свойству надежность.

34. Провести сравнительный анализ Linux и Windowsпо свойству защиты.

35. Постройте обобщенную схему двухуровневой операционной системы.

36.Постройте обобщенную схему алгоритма работы планировщика памяти.

37. Провести сравнительный анализ по основным способам организации вычислительного процесса операционных систем Linux и Windows.

38. Постройте граф существования процесса для ОС общего назначения.

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

40. Постройте многоочередную схему управления процессами.

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

**Операционная система (ОС)–** это комплекс взаимосвязанных системных программ для организации взаимодействия пользователя с компьютером и выполнения всех других программ.

|  |  |
| --- | --- |
|  |  |

*Аппаратура* — комплекс, совокупность аппаратов (завершённой совокупность частей или элементов для выполнения какой-либо функции).

*Средства аппаратной поддержки ОС* – это средства, которые прямо участвуют в организации вычислительных процессов ОС: средства поддержки привилегированного режима, систему прерываний, средства переключения контекстов процессов, средства защиты областей памяти и т. п.

*Машиннозависимые модули ОС* - образует модули, в которых отражается специфика аппаратной платформы компьютера.

*Базовые механизмы ядра ос* - этот слой модулей выполняет наиболее примитивные операции ядра: программное переключение контекстов процессов, диспетчерскую 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.

*Менеджеры ресурсов*, являющиеся неотъемлемыми частями обычного ядра − файловая система, подсистемы управления виртуальной памятью и процессами, менеджер безопасности.

*Файловая система* - порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов.

# *Подсистема управления памятью* **-** осуществляет распределение имеющейся физической памяти между всеми существующими в системе в данный момент процессами, загрузку кодов и данных процессов в отведенные им области памяти, настройку адресно-зависимых частей кодов процесса на физические адреса выделенной области, а также защиту областей памяти каждого процесса.

*Интерфейс системных вызовов API* - это определенный набор протоколов, подпрограмм и инструментов для создания программных приложений.

*Утилиты, системные программы*- вспомогательная компьютерная программа в составе общего программного обеспечения для выполнения специализированных типовых задач, связанных с работой оборудования и операционной системы (ОС).

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

См. вопрос 21.

**3. Приведите основные стратегии планировщиков.**

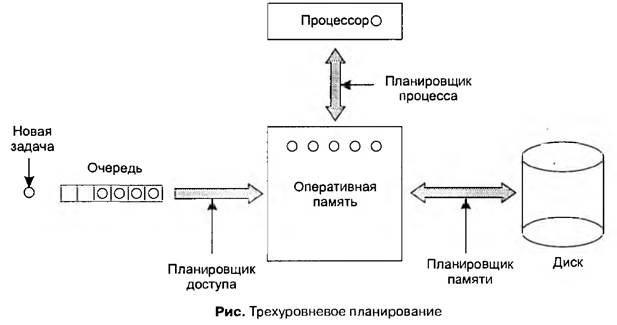
Пакетные системы:

1) *«Первый пришел — первый обслуживается»* (FIFO — first in, first out, или FCFS — first come, first served). FCFS является наиболее простой стратегией планирования процессов и заключается в том, что процессор передается тому процессу, который раньше всех других его запросил. Когда процесс попадает в очередь готовых процессов, УТП (РСВ) присоединяется к хвосту очереди. Среднее время ожидания для стратегии FCFS часто весьма велико и зависит от порядка поступления процессов в очередь готовых процессов. Стратегии FCFS присущ так называемый «эффект конвоя». В том случае, когда в компьютере имеется один большой процесс и несколько малых, то все процессы собираются в начале очереди готовых процессов, а затем в очереди к оборудованию. Таким образом, «эффект конвоя» приводит к снижению загруженности как процессора, так и периферийного оборудования.

2) *Кратчайшая задача первая* - SJF — Shortest Job First. Одним из методов борьбы с «эффектом конвоя» является стратегия, позволяющая процессу из очереди выполняться первым. Стратегия SJF снижает время ожидания очереди. Наибольшая трудность в практической реализации SJF заключается в невозможности заранее определить величину времени последующего обслуживания. Поэтому стратегия SJF часто применяется в долгосрочных планировщиках, обслуживающих пакетный режим. В этом случае вместо величины времени последующего обслуживания используется допустимое максимальное время выполнения задания, которое программист должен специфицировать перед отправкой задания в пакет.

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

4) *Трехуровневое планирование* - по мере поступления в систему новые задачи сна­чала помещаются в очередь, хранящуюся на диске. Впускной **планировщик доступа** выбирает задание и передает его системе. Остальные задания остаются в очереди.



Как только задание попало в систему, для него будет создан соответствующий процесс, и он может тут же вступить в борьбу за доступ к процессору. Тем не менее возможна ситуация, когда процессов слишком много и они все в памяти не помещаются, тогда некоторые из них будут выгружены на диск. Второй уровень планирования определяет, какие процессы можно хранить в памяти, а какие — на диске. Этим занимается **планировщик памяти.** Планировщик памяти периодически просматривает процессы, находящиеся на диске, чтобы решить, какой из них переместить в память. Третий уровень планирования отвечает за доступ процессов, находящихся в со­стоянии готовности, к процессору.

Интерактивные системы:

***1) Циклическое планирование* -** одним из наиболее старых, простых, справедливых и часто используемых является алгоритм циклического планирования. Каждому процессу предоставля­ется некоторый интервал времени процессора, так называемый квант времени. Если к концу кванта времени процесс все еще работает, он прерывается, а управ­ление передается другому процессу. Разумеется, если процесс блокируется или прекращает работу раньше, переход управления происходит в этот момент. Реа­лизация циклического планирования проста. Планировщику нужно всего лишь поддерживать список процессов в состоянии готовности. Когда процесс исчерпал свой лимит времени, он отправляется в конец списка.Единственным интересным моментом этого алгоритма является длина кванта. Переключение с одного процесса на другой занимает некоторое время — необхо­димо сохранить и загрузить регистры и карты памяти, обновить таблицы и списки, сохранить и перезагрузить кэш памяти и т. п. Вывод можно сформулировать следующим образом: слишком малый квант приведет к частому переключению процессов и небольшой эффективности, но слишком большой квант может привести к медленному реагированию на корот­кие интерактивные запросы. Значение кванта около 2.0 -5.0 мс часто является ра­зумным компромиссом.

*2) Приоритетное планирование* **-** в циклическом алгоритме планирования есть важное допущение о том, что все процессы равнозначны. В ситуации компьютера с большим числом пользователей это может быть не так. Например, в университете прежде всего должны обслужи­ваться деканы, затем профессора, секретари, уборщицы и лишь потом студенты. Необходимость принимать во внимание подобные внешние факторы приводит к приоритетному планированию. Основная идея проста: каждому процессу присваи­вается приоритет, и управление передается готовому к работе процессу с самым высоким приоритетом.

*3) Самый короткий процесс – следующий.*Поскольку, алгоритм «Кратчайшая задача — первая» минимизирует среднее оборот­ное время в системах пакетной обработки, хотелось бы использовать его и в инте­рактивных системах. В известной степени это возможно. Интерактивные процес­сы чаще всего следуют схеме «ожидание команды, исполнение команды, ожида­ние команды, исполнение команды...» Если рассматривать выполнение каждой команды как отдельную задачу, можно минимизировать общее среднее время от­клика, запуская первой самую короткую задачу. Единственная проблема состоит в том, чтобы понять, какой из ожидающих процессов самый короткий.

*4) Гарантированное планирование* **-** Принципиально другим подходом к планированию является предоставление пользователям реальных обещаний и затем их выполнение. Вот одно обещание, которое легко произнести и легко выполнить: если вместе с вами процессором пользуются n пользователей, вам будет предоставлено 1/n мощности процессора. И в системе с одним пользователем и n запущенными процессорами каждому достанется 1/n циклов процессора. Чтобы выполнить это обещание, система должна отслеживать распределение процессора между процессами с момента создания каждого процесса. Затем система рассчитывает количество ресурсов процессора, на которое процесс имеет право, например время с момента создания, деленное на n. Теперь можно сосчитать отношение времени, предоставленного процессу, к времени, на которое он имеет право. Полученное значение 0,5 означает, что процессу выделили только полови­ну положенного, а 2,0 означает, что процессу досталось в два раза больше, чем положено. Затем запускается процесс, у которого это отношение наименьшее, пока оно не станет больше, чем у его ближайшего соседа.

***5) Лотерейное планирование* -** в основе алгоритма лежит раздача процессам лотерейных билетов на доступ к различным ресурсам, в том числе и к процессору. Когда планировщику необходимо принять решение, выбирается случайным образом лотерейный билет, и его обладатель получает доступ к ресурсу. Что касается доступа к процессору, «лотерея» может происходить 50 раз в секунду, и победитель получает 20 мс времени процессора. Более важным процессам можно раздать дополнительные билеты, чтобы увеличить вероятность выигрыша. Если всего 100 билетов и 20 из них находятся у одного процесса, то ему достанется 20 % времени процессора. В отличие от приоритетного планировщика, в котором очень трудно оценить, что означает, скажем, приоритет 40, в лотерейном планировании все очевидно. Каждый процесс получит процент ресурсов, примерно равный проценту имеющихся у него билетов. Лотерейное планирование характеризуется несколькими интересными свойствами. Например, если при создании процессу достается несколько билетов, то уже в следующей лотерее его шансы на выигрыш пропорциональны количеству билетов. Взаимодействующие процессы могут при необходимости обмениваться билетами. Так, если клиентский процесс посылает сообщение серверному процессу и затем блокируется, он может передать все свои билеты серверному процессу, что­бы увеличить шанс запуска сервера. Когда серверный процесс заканчивает работу, он может вернуть все билеты обратно.

***6) 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.

Системы реального времени:

***Планирование в системах реального времени -*** в системах реального времени существенную роль играет время. Чаще всего одно или несколько внешних физических устройств генерируют входные сигналы, и компьютер должен адекватно на них реагировать в течение заданного промежутка времени.Системы реального времени делятся на ***жесткие системы реального времени***, что означает наличие жестких сроков для каждой задачи (в них обязательно надо укладываться), и ***мягкие системы реального времени***, в которых нарушения временного графика нежелательны, но допустимы. В обоих случаях реализуется раз­деление программы на несколько процессов, каждый из которых предсказуем. Эти процессы чаще всего бывают короткими и завершают свою работу в течение секунды. Когда появляется внешний сигнал, именно планировщик должен обеспечить соблюдение графика.Внешние события, на которые система должна реагировать, можно разделить на *периодические*(возникающие через регулярные интервалы времени) и *непериодические*(возникающие непредсказуемо). Возможно наличие нескольких пери­одических потоков событий, которые система должна обрабатывать. В зависимости от времени, затрачиваемого на обработку каждого из событий, может оказаться, что система не в состоянии своевременно обработать все события.

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

**Подкачка,** или **своппинг** (**swapping** - обмен), - это процесс выгрузки редко используемых областей виртуального адресного пространства программы и/или всей программы на диск или другое устройство внешней памяти. Такая внешняя память всегда намного дешевле оперативной памяти, хотя и намного медленнее. Свопинг представляет собой частный случай виртуальной памяти и, следователь­но, более простой в реализации способ совместного использования оператив­ной памяти и диска. Однако подкачке свойственна избыточность: когда ОС ре­шает активизировать процесс, для его выполнения, как правило, не требуется за­гружать в оперативную память все его сегменты полностью — достаточно загрузить небольшую часть кодового сегмента с подлежащей выполнению инст­рукцией и частью сегментов данных, с которыми работает эта инструкция, а так­же отвести место под сегмент стека. Аналогично при освобождении памяти для загрузки нового процесса очень часто вовсе не требуется выгружать другой про­цесс на диск целиком, достаточно вытеснить на диск только часть его образа. Перемещение избыточной информации замедляет работу системы, а также при­водит к неэффективному использованию памяти. Кроме того, системы, поддер­живающие свопинг, имеют еще один очень существенный недостаток: они не способны загрузить для выполнения процесс, виртуальное адресное пространст­во которого превышает имеющуюся в наличии свободную память.

**5. Сформулируйте основные функции операционных систем.**

ОС выполняет две группы функций:

- предоставление пользователям и программистам вместо реальной аппаратуры компьютера расширенной виртуальной машины, с которой удобней работать и легче программировать;

- повышение эффективности использования компьютера путем рационального управления его ресурсами.

## Основные функции ОС:

## управление устройствами компьютера (ресурсами), т.е. согласованная работа всех аппаратных средств ПК: стандартизованный доступ к периферийным устройствам, управление оперативной памятью и др.

## управление процессами, т.е. выполнение программ и их взаимодействие с устройствами компьютера.

## управление доступом к данным на энергонезависимых носителях (таких как жесткий диск, компакт-диск и т.д.), как правило, с помощью файловой системы.

## ведение файловой структуры.

## пользовательский интерфейс, т.е. диалог с пользователем.

## 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).

**6. Приведите классификацию ресурсов вычислительной системы.**

*По форме реализации:*

- аппаратные ресурсы (Hard);

- программные ресурсы (Soft);

- информационные ресурсы.

*По способу выделения ресурса различают:*

- неделимые ресурсы – предоставляются процессу в полное распоряжение;

- делимые ресурсы – предоставляются процессу в соответствии с запросом на требуемое количество ресурса.

*По реальности существования различают:*

- физический ресурс – реально существует и при распределении обладает всеми присущими ему свойствами;

- виртуальный ресурс – программно-аппаратная модель физического ресурса.

*По месту размещения ресурса различают:*

- локальные ресурсы – принадлежат автономному компьютеру;

- удаленные ресурсы – принадлежат рабочим станциям или серверам, входящим в состав сети.

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

Первое поколение.

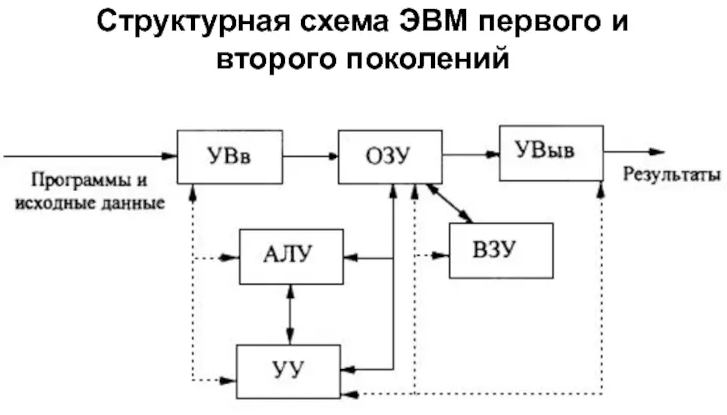
40-е годы. Первые цифровые вычислительные машины без ОС. Организация вычислительного процесса решается программистом с пульта управления.

Второе поколение.

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

Пакетный режим

Необходимость оптимального использования 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

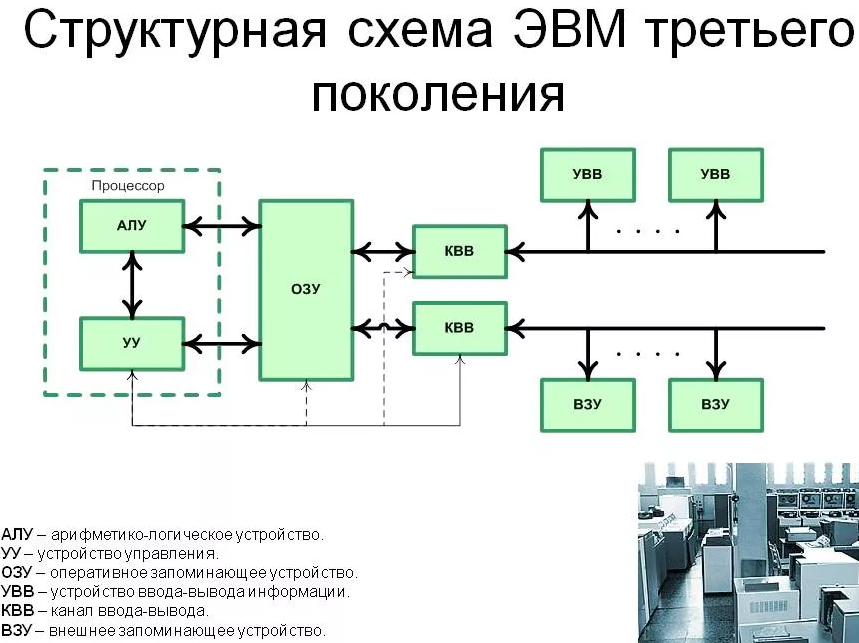
Применение универсальных компьютеров для управления производственными процессами потребовало реализации «реального масштаба времени» («реального времени») — синхронизации исполнения программ с внешними физическими процессами.

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.



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

Потребность в синхронизации потоков возникает только в мультипрограммной операционной системе и связана с совместным использованием аппаратных и информационных ресурсов вычислительной системы. Синхронизация необходи­ма для исключения гонок и тупиков при обмене данными между потоками, раз­делении данных, при доступе к процессору и устройствам ввода-вывода. Выполнение потока в мультипрограммной среде всегда имеет асинхронный ха­рактер. Очень сложно с полной определенностью сказать, на каком этапе выпол­нения будет находиться процесс в определенный момент времени. Даже в однопрограммном режиме не всегда можно точно оценить время выполнения задачи. Это время во многих случаях существенно зависит от значения исходных дан­ных, которые влияют на количество циклов, направления разветвления програм­мы, время выполнения операций ввода-вывода и т. п. Так как исходные данные в разные моменты запуска задачи могут быть разными, то и время выполнения от­дельных этапов и задачи в целом является весьма неопределенной величиной.

Еще более неопределенным является время выполнения программы в мульти­программной системе. Моменты прерывания потоков, время нахождения их в очередях к разделяемым ресурсам, порядок выбора потоков для выполнения — все эти события являются результатом стечения многих обстоятельств и могут быть интерпретированы как случайные. В лучшем случае можно оценить веро­ятностные характеристики вычислительного процесса, например вероятность его завершения за данный период времени.

Таким образом, потоки в общем случае (когда программист не предпринял спе­циальных мер по их синхронизации) протекают независимо, асинхронно друг другу. 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.

Любое взаимодействие процессов или потоков связано с их *синхронизацией,*ко­торая заключается в согласовании их скоростей путем приостановки потока до наступления некоторого события и последующей его активизации при наступ­лении этого события. Синхронизация лежит в основе любого взаимодействия потоков, связано ли это взаимодействие с разделением ресурсов или с обменом данными. Например, поток-получатель должен обращаться за данными только после того, как они помещены в буфер потоком-отправителем. Если же поток-получатель обратился к данным до момента их поступления в буфер, то он дол­жен быть приостановлен.

При совместном использовании аппаратных ресурсов синхронизация также со­вершенно необходима. Когда, например, активному потоку требуется доступ к последовательному порту, а с этим портом в монопольном режиме работает дру­гой поток, находящийся в данный момент в состоянии ожидания, то ОС приос­танавливает активный поток и не активизирует его до тех пор, пока нужный ему порт не освободится. Часто нужна также синхронизация с событиями, внешними по отношению к вычислительной системе, например реакции на нажатие комби­нации клавиш Ctrl+C. Ежесекундно в системе происходят сотни событий, связанных с распределением и освобождением ресурсов, и ОС должна иметь надежные и производительные средства, которые бы позволяли ей синхронизировать потоки с происходящими в системе событиями.

Для синхронизации потоков прикладных программ программист может исполь­зовать как собственные средства и приемы синхронизации, так и средства опера­ционной системы. Например, два потока одного прикладного процесса могут ко­ординировать свою работу с помощью доступной для них обоих глобальной логической переменной, которая устанавливается в единицу при осуществлении некоторого события, например выработки одним потоком данных, нужных для продолжения работы другого. Однако во многих случаях более эффективными или даже единственно возможными являются средства синхронизации, предос­тавляемые операционной системой в форме системных вызовов. Так, потоки, принадлежащие разным процессам, не имеют возможности вмешиваться каким-либо образом в работу друг друга. Без посредничества операционной системы они не могут приостановить друг друга или оповестить о произошедшем собы­тии. Средства синхронизации используются операционной системой не только для синхронизации прикладных процессов, но и для ее внутренних нужд.

Обычно разработчики операционных систем предоставляют в распоряжение при­кладных и системных программистов широкий спектр средств синхронизации. Эти средства могут образовывать иерархию, когда на основе более простых средств строятся более сложные, а также быть функционально специализированными, например средства для синхронизации потоков одного процесса, средства для синхронизации потоков разных процессов при обмене данными и т. д. Часто функциональные возможности разных системных вызовов синхронизации пере­крываются, так что для решения одной задачи программист может воспользо­ваться несколькими вызовами в зависимости от своих личных предпочтений.

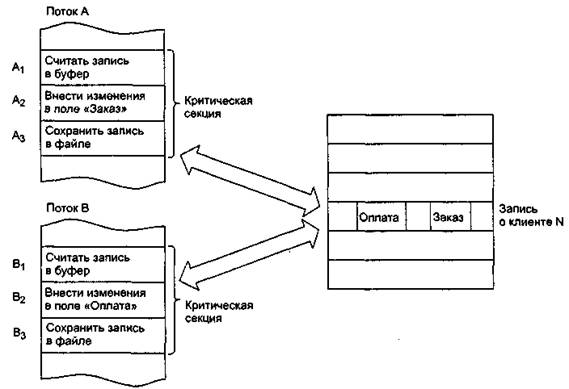
*Необходимость синхронизации и гонки*

Пренебрежение вопросами синхронизации в многопоточной системе может при­вести к неправильному решению задачи или даже к краху системы. Рассмотрим, например (рис. 4.16), задачу ведения базы данных клиентов некоторого предпри­ятия. Каждому клиенту отводится отдельная запись в базе данных, в которой среди прочих полей имеются поля Заказ и Оплата. Программа, ведущая базу дан­ных, оформлена как единый процесс, имеющий несколько потоков, в том числе поток А, который заносит в базу данных информацию о заказах, поступивших от клиентов, и поток В, который фиксирует в базе данных сведения об оплате кли­ентами выставленных счетов. Оба эти потока совместно работают над общим файлом базы данных, используя однотипные алгоритмы, включающие три шага.

1. Считать из файла базы данных в буфер запись о клиенте с заданным иденти­фикатором.

2. Внести новое значение в поле Заказ (для потока А) или Оплата (для потока В).

3. Вернуть модифицированную запись в файл базы данных.



**Рис. 4.16.**Возникновение гонок при доступе к разделяемым данным

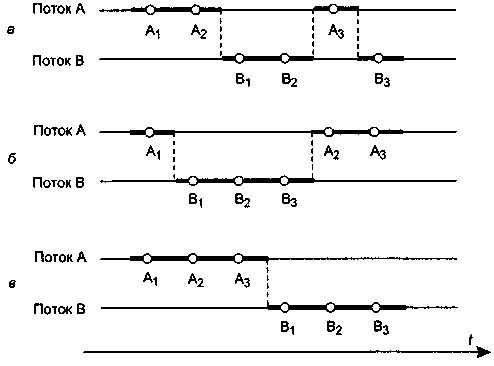
Обозначим соответствующие шаги для потока А как Al, A2 и A3, а для потока В как 1, В2 и ВЗ. Предположим, что в некоторый момент поток А обновляет поле Заказ записи о клиенте N. Для этого он считывает эту запись в свой буфер (шаг А1), модифицирует значение поля Заказ (шаг А2), но внести запись в базу данных (шаг A3) не успевает, так как его выполнение прерывается, например, вследствие завершения кванта времени.

Предположим также, что потоку В также потребовалось внести сведения об оп­лате относительно того же клиента N. Когда подходит очередь потока В, он успе­вает считать запись в свой буфер (шаг В1) и выполнить обновление поля Оплата (шаг В2), а затем прерывается. Заметим, что в буфере у потока В находится за­пись о клиенте N, в которой поле Заказ имеет прежнее, не измененное значение.

Когда в очередной раз управление будет передано потоку А, то он, продолжая свою работу, запишет запись о клиенте N с модифицированным полем Заказ в базу данных (шаг A3). После прерывания потока А и активизации потока В по­следний запишет в базу данных поверх только что обновленной записи о клиен­те N свой вариант записи, в которой обновлено значение поля Оплата. Таким об­разом, в базе данных будут зафиксированы сведения о том, что клиент N произвел оплату, но информация о его заказе окажется потерянной (рис. 4.17, а).

Сложность проблемы синхронизации кроется в нерегулярности возникающих ситуаций. Так, в предыдущем примере можно представить и другое развитие со­бытий: могла быть потеряна информация не о заказе, а об оплате (рис. 4.17, *6)*

или, напротив, все исправления были успешно внесены (рис. 4.17, *в).*Все опреде­ляется взаимными скоростями потоков и моментами их прерывания. Поэтому отладка взаимодействующих потоков является сложной задачей. Ситуации, по­добные той, когда два или более потоков обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей потоков, называются *гонками.*



**Рис. 4.17.**Влияние относительных скоростей потоков на результат решения задачи

*Критическая секция*

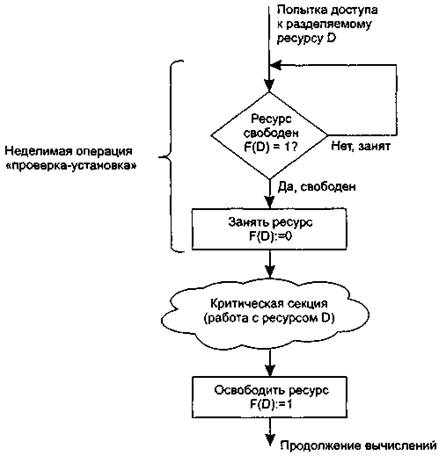
Важным понятием синхронизации потоков является понятие «критической сек­ции» программы. *Критическая секция*— это часть программы, результат выпол­нения которой может непредсказуемо меняться, если переменные, относящиеся к этой части программы, изменяются другими потоками в то время, когда вы­полнение этой части еще не завершено. Критическая секция всегда определяется по отношению к определенным *критическим данным,*при несогласованном из­менении которых могут возникнуть нежелательные эффекты. В предыдущем при­мере такими критическими данными являлись записи файла базы данных. Во всех потоках, работающих с критическими данными, должна быть определена критическая секция. Заметим, что в разных потоках критическая секция состоит в общем случае из разных последовательностей команд.

Чтобы исключить эффект гонок по отношению к критическим данным, необхо­димо обеспечить, чтобы в каждый момент времени в критической секции, свя­занной с этими данными, находился только один поток. При этом неважно, нахо­дится этот поток в активном или в приостановленном состоянии. Этот прием называют *взаимным исключением.*Операционная система использует разные способы реализации взаимного исключения. Некоторые способы пригодны для взаимного исключения при вхождении в критическую секцию только потоков одного процесса, в то время как другие могут обеспечить взаимное исключение и для потоков разных процессов.

Самый простой и в то же время самый неэффективный способ обеспечения вза­имного исключения состоит в том, что операционная система позволяет потоку запрещать любые прерывания на время его нахождения в критической секции. Однако этот способ практически не применяется, так как опасно доверять управ­ление системой пользовательскому потоку — он может надолго занять процес­сор, а при крахе потока в критической секции крах потерпит вся система, потому что прерывания никогда не будут разрешены.

*Блокирующие переменные*

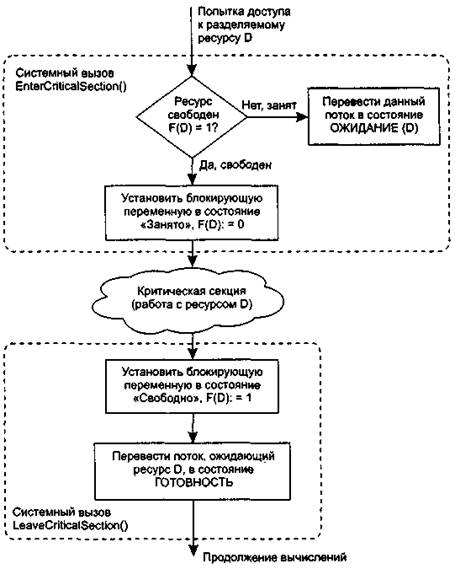
Для синхронизации потоков одного процесса прикладной программист может использовать глобальные *блокирующие переменные.*С этими переменными, к ко­торым все потоки процесса имеют прямой доступ, программист работает, не об­ращаясь к системным вызовам ОС.



**Рис. 4.18.**Реализация критических секций с использованием блокирующих переменных

Каждому набору критических данных ставится в соответствие двоичная пере­менная, которой поток присваивает значение 0, когда он входит в критическую секцию, и значение 1, когда он ее покидает. На рис. 4.18 показан фрагмент алго­ритма потока, использующего для реализации взаимного исключения доступа к критическим данным D блокирующую переменную F(D). Перед входом в критическую секцию поток проверяет, не работает ли уже какой-нибудь поток с дан­ными D. Если переменная F(D) установлена в 0, то данные заняты и проверка циклически повторяется. Если же данные свободны (F(D) = 1), то значение пере­менной F(D) устанавливается в 0 и поток входит в критическую секцию. После того как поток выполнит все действия с данными D, значение переменной F(D) снова устанавливается равным 1. Блокирующие переменные могут использоваться не только при доступе к разде­ляемым данным, но и при доступе к разделяемым ресурсам любого вида. Если все потоки написаны с учетом вышеописанных соглашений, то взаимное исключение гарантируется. При этом потоки могут быть прерваны операци­онной системой в любой момент и в любом месте, в том числе в критической секции. Однако следует заметить, что одно ограничение на прерывания все же имеется. Нельзя прерывать поток между выполнением операций проверки и установки блокирующей переменной. Поясним это. Пусть в результате проверки перемен­ной поток определил, что ресурс свободен, но сразу после этого, не успев уста­новить переменную в 0, был прерван. За время его приостановки другой поток занял ресурс, вошел в свою критическую секцию, но также был прерван, не за­вершив работы с разделяемым ресурсом. Когда управление было возвращено первому потоку, он, считая ресурс свободным, установил признак занятости и начал выполнять свою критическую секцию. Таким образом, был нарушен прин­цип взаимного исключения, что потенциально может привести к нежелательным последствиям. Во избежание таких ситуаций в системе команд многих компью­теров предусмотрена единая, неделимая команда анализа и присвоения значения логической переменной (например, команды ВТС, BTR и BTS процессора Pentium). При отсутствии такой команды в процессоре соответствующие действия должны реализовываться специальными системными примитивами', которые бы запре­щали прерывания на протяжении всей операции проверки и установки. Реализация взаимного исключения описанным выше способом имеет существен­ный недостаток: в течение времени, когда один поток находится в критической секции, другой поток, которому требуется тот же ресурс, получив доступ к про­цессору, будет непрерывно опрашивать блокирующую переменную, бесполезно тратя выделяемое ему процессорное время, которое могло бы быть использовано для выполнения какого-нибудь другого потока. Для устранения этого недостат­ка во многих ОС предусматриваются специальные системные вызовы для рабо­ты с критическими секциями.

На рис. 4.19 показано, как с помощью этих функций реализовано взаимное ис­ключение в операционной системе Windows NT. Перед тем как начать измене­ние критических данных, поток выполняет системный вызов EnterCriticalSection(). В рамках этого вызова сначала выполняется, как и в предыдущем случае, проверка блокирующей переменной, отражающей состояние критического ресур­са. Если системный вызов определил, что ресурс занят (F(D) = 0), он в отличие от предыдущего случая не выполняет циклический опрос, а переводит поток в состояние ожидания (D) и делает отметку о том, что данный поток должен быть активизирован, когда соответствующий ресурс освободится. Поток, который в это время использует данный ресурс, после выхода из критической секции дол­жен выполнить системную функцию LeaveCriticalSectionO, в результате чего блокирующая переменная принимает значение, соответствующее свободному состоянию ресурса (F(D) = 1), а операционная система просматривает очередь ожидающих этот ресурс потоков и переводит первый поток из очереди в состоя­ние готовности.



**Рис. 4.19.**Реализация взаимного исключения с использованием системных функций входа в критическую секцию и выхода из нее

Таким образом исключается непроизводительная потеря процессорного времени на циклическую проверку освобождения занятого ресурса. Однако в тех случаях, когда объем работы в критической секции небольшой и существует высокая ве­роятность в очень скором доступе к разделяемому ресурсу, более предпочтитель­ным может оказаться использование блокирующих переменных. Действительно, в такой ситуации накладные расходы ОС по реализации функции входа в крити­ческую секцию и выхода из нее могут превысить полученную экономию.

*Семафоры*

Обобщением блокирующих переменных являются так называемые *семафоры Дийкстры.*Вместо двоичных переменных Дийкстра (Dijkstra) предложил исполь­зовать переменные, которые могут принимать целые неотрицательные значения. Такие переменные, используемые для синхронизации вычислительных процес­сов, получили название семафоров. Для работы с семафорами вводятся два примитива, традиционно обозначаемых Р и V. Пусть переменная S представляет собой семафор. Тогда действия V(S) и P(S) определяются следующим образом.

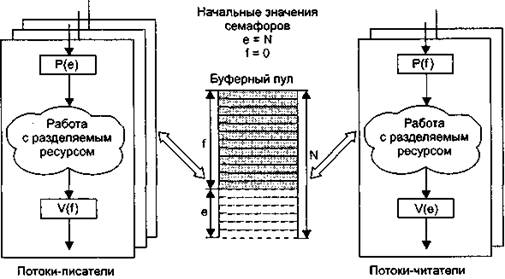
\* V(S): переменная S увеличивается на 1 единым действием. Выборка, наращи­вание и запоминание не могут быть прерваны. К переменной S нет доступа другим потокам во время выполнения этой операции.

\* P(S): уменьшение S на 1, если это возможно. Если S=0 и невозможно умень­шить S, оставаясь в области целых неотрицательных значений, то в этом случае поток, вызывающий операцию Р, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также являются неделимой операцией.

Никакие прерывания во время выполнения примитивов V и Р недопустимы. В частном случае, когда семафор S может принимать только значения 0 и 1, он превращается в блокирующую переменную, которую по этой причине часто на­зывают двоичным семафором. Операция Р заключает в себе потенциальную воз­можность перехода потока, который ее выполняет, в состояние ожидания, в то время как операция V может при некоторых обстоятельствах активизировать дру­гой поток, приостановленный операцией Р. Рассмотрим использование семафоров на классическом примере взаимодействия двух выполняющихся в режиме мультипрограммирования потоков, один из ко­торых пишет данные в буферный пул, а другой считывает их из буферного пула. Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. В общем случае поток-писатель и поток-читатель могут иметь раз­личные скорости и обращаться к буферному пулу с переменой интенсивностью. В один период скорость записи может превышать скорость чтения, в другой — наоборот. Для правильной совместной работы поток-писатель должен приоста­навливаться, когда все буферы оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, поток-читатель должен приос­танавливаться, когда все буферы пусты, и активизироваться при появлении хотя бы одной записи.

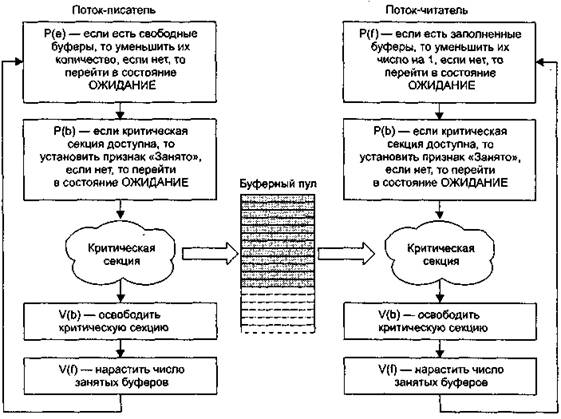
Введем два семафора: е — число пустых буферов, и f — число заполненных буфе­ров, причем в исходном состоянии е = N, a f = 0. Тогда работа потоков с общим буферным пулом может быть описана следующим образом (рис. 4.20).

Поток-писатель прежде всего выполняет операцию Р(е), с помощью которой он проверяет, имеются ли в буферном пуле незаполненные буферы. В соответствии с семантикой операции Р, если семафор е равен 0 (то есть свободных буферов в данный момент нет), то поток-писатель переходит в состояние ожидания. Если же значением е является положительное число, то он уменьшает число свободных буферов, записывает данные в очередной свободный буфер и после этого наращи­вает число занятых буферов операцией V(f). Поток-читатель действует анало­гичным образом, с той разницей, что он начинает работу с проверки наличия заполненных буферов, а после чтения данных наращивает количество свободных буферов.



**Рис. 4.20.**Использование семафоров для синхронизации потоков

В данном случае предпочтительнее использовать семафоры вместо блокирующих переменных. Действительно, критическим ресурсом здесь является буферный пул, который может быть представлен как набор идентичных ресурсов — отдель­ных буферов, а значит, с буферным пулом могут работать сразу несколько пото­ков, и именно столько, сколько буферов в нем содержится. Использование дво­ичной переменной не позволяет организовать доступ к критическому ресурсу более чем одному потоку. Семафор же решает задачу синхронизации более гиб­ко, допуская к разделяемому пулу ресурсов заданное количество потоков. Так, в нашем примере с буферным пулом могут работать максимум N потоков, часть из которых может быть «писателями», а часть — «читателями». Таким образом, семафоры позволяют эффективно решать задачу синхронизации доступа к ресурсным пулам, таким, например, как набор идентичных в функцио­нальном назначении внешних устройств (модемов, принтеров, портов), или на­бор областей памяти одинаковой величины, или информационных структур. Во всех этих и подобных им случаях с помощью семафоров можно организовать доступ к разделяемым ресурсам сразу нескольких потоков.



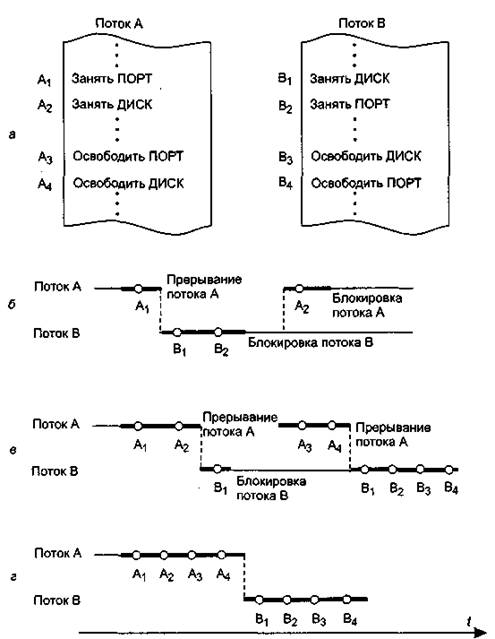
**Рис. 4.21.**Использование двоичного семафора

*Тупики*

Приведенный выше пример позволяет также проиллюстрировать еще одну проблему синхронизации — *взаимные блокировки,*называемые также *дедлоками (deadlocks), клинчами (clinch),*или *тупиками.*Покажем, что если переставить мес­тами операции Р(е) и Р(b) в потоке-писателе, то при некотором стечении обстоя­тельств эти два потока могут взаимно блокировать друг друга.

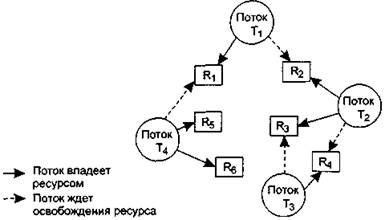
Итак, пусть поток-писатель начинает свою работу с проверки доступности кри­тической секции — операции Р(b), и пусть он первым войдет в критическую сек­цию. Выполняя операцию Р(е), он может обнаружить отсутствие свободных бу­феров и перейти в состояние ожидания. Как уже было показано, из этого состоя­ния его может вывести только поток-читатель, который возьмет очередную за­пись из буфера. Но поток-читатель не сможет этого сделать, так как для этого ему потребуется войти в критическую секцию, вход в которую заблокирован по­током-писателем. Таким образом, ни один из этих потоков не может завершить начатую работу и возникнет тупиковая ситуация, которая не может разрешиться без внешнего воздействия.

Рассмотрим еще один пример тупика. Пусть двум потокам, принадлежащим разным процессам и выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например принтер и последовательный порт. Такая ситуация может возникнуть, например, во время работы приложения, задачей кото­рого является распечатка информации, поступающей по модемной связи. На рис. 4.22, *а*показаны фрагменты соответствующих программ. Поток А запра­шивает сначала принтер, а затем порт, а поток В запрашивает устройства в обрат­ном порядке. Предположим, что после того, как ОС назначила принтер потоку А и установила связанную с этим ресурсом блокирующую переменную, поток А был прерван. Управление получил поток В, который сначала выполнил запрос на получение СОМ-порта, затем при выполнении следующей команды был забло­кирован, так как принтер оказался уже занятым потоком А. Управление снова по­лучил поток А, который в соответствии со своей программой сделал попытку за­нять порт и был заблокирован, поскольку порт уже выделен потоку В. В таком положении потоки А и В могут находиться сколь угодно долго. В зависимости от соотношения скоростей потоков они могут либо взаимно бло­кировать друг друга (рис. 4.22, *б),*либо образовывать очереди к разделяемым ре­сурсам (рис. 4.22, б), либо совершенно независимо использовать разделяемые ре­сурсы (рис. 4.22, *г).*



**Рис. 4.22.**Возникновение взаимных блокировок при выполнении программы

Тупиковые ситуации надо отличать от простых очередей, хотя те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: поток приостанавлива­ется и ждет освобождения ресурса. Однако очередь — это нормальное явление, неотъемле­мый признак высокого коэффициента использования ресурсов при случайном поступле­нии запросов. Очередь появляется тогда, когда ресурс недоступен в данный момент, но освободится через некоторое время, позволив потоку продолжить выполнение. Тупик же, что видно из его названия, является в некотором роде неразрешимой ситуацией. Необхо­димым условием возникновения тупика является потребность потока сразу в нескольких ресурсах. В рассмотренных примерах тупик был образован двумя потоками, но взаимно блокировать друг друга может и большее число потоков. На рис. 2.23 показано такое распределение ресурсов Ri между несколькими потоками Tj, которое при­вело к возникновению взаимных блокировок. Стрелки обозначают потребность потока в ресурсах. Сплошная стрелка означает, что соответствующий ресурс был выделен потоку, а пунктирная стрелка соединяет поток с тем ресурсом, который необходим, но не может быть пока выделен, поскольку занят другим потоком. Например, потоку Т1 для выполнения работы необходимы ресурсы R1 и R2, из ко­торых выделен только один — R1, а ресурс R2 удерживается потоком Т2. Ни один из четырех показанных на рисунке потоков не может продолжить свою работу, так как не имеет всех необходимых для этого ресурсов. Невозможность потоков завершить начатую работу из-за возникновения вза­имных блокировок снижает производительность вычислительной системы. По­этому проблеме предотвращения тупиков уделяется большое внимание. На тот случай, когда взаимная блокировка все же возникает, система должна предоста­вить администратору-оператору средства, с помощью которых он смог бы распо­знать тупик, отличить его от обычной блокировки из-за временной недоступности ресурсов. И наконец, если тупик диагностирован, то нужны средства для снятия взаимных блокировок и восстановления нормального вычислительного процесса.



**Рис. 4.23.**Взаимная блокировка нескольких потоков

Тупики могут быть предотвращены на стадии написания программ, то есть про­граммы должны быть написаны таким образом, чтобы тупик не мог возникнуть при любом соотношении взаимных скоростей потоков. Так, если бы в примере, показанном на рис. 4.22, поток А и поток В запрашивали ресурсы в одинако­вой последовательности, то тупик был бы в принципе невозможен. Другой, бо­лее гибкий подход к предотвращению тупиков заключается в том, что ОС каж­дый раз при запуске задач анализирует их потребности в ресурсах и определяет, может ли в данной мультипрограммной смеси возникнуть тупик. Если да, то за­пуск новой задачи временно откладывается. ОС может также использовать опре­деленные правила при назначении ресурсов потокам, например, ресурсы могут выделяться операционной системой в определенной последовательности, общей для всех потоков. В тех же случаях, когда тупиковую ситуацию не удалось предотвратить, важно быстро и точно ее распознать, поскольку блокированные потоки не выполняют никакой полезной работы. Если тупиковая ситуация образована множеством по­токов, занимающих массу ресурсов, распознавание тупика является нетривиаль­ной задачей. Существуют формальные, программно-реализованные методы рас­познавания тупиков, основанные на ведении таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ этих таблиц позволяет обнаружить взаимные блокировки. Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные потоки. Можно снять только часть из них, освободив ресурсы, ожидаемые остальными потоками, можно вернуть некоторые потоки в область под­качки, можно совершить «откат» некоторых потоков до так называемой контроль­ной точки, в которой запоминается вся информация, необходимая для восстановле­ния выполнения программы с данного места. Контрольные точки расставляются в программе в тех местах, после которых возможно возникновение тупика.

***Синхронизирующие объекты ОС***

Рассмотренные выше механизмы синхронизации, основанные на использовании глобальных переменных процесса, обладают существенным недостатком — они не подходят для синхронизации потоков разных процессов. В таких случаях опе­рационная система должна предоставлять потокам системные объекты синхро­низации, которые были бы видны для всех потоков, даже если они принадлежат разным процессам и работают в разных адресных пространствах. Примерами таких синхронизирующих объектов ОС являются системные сема­форы, мьютексы, события, таймеры и другие — их набор зависит от конкретной ОС, которая создает эти объекты по запросам процессов. Чтобы процессы могли разделять синхронизирующие объекты, в разных ОС используются разные методы. Некоторые ОС возвращают указатель на объект. Этот указатель может быть до­ступен всем родственным процессам, наследующим характеристики общего ро­дительского процесса. В других ОС процессы в запросах на создание объектов синхронизации указывают имена, которые должны быть им присвоены. Далее эти имена используются разными процессами для манипуляций объектами син­хронизации. В таком случае работа с синхронизирующими объектами подобна ра­боте с файлами. Их можно создавать, открывать, закрывать, уничтожать. Кроме того, для синхронизации могут быть использованы такие «обычные» объ­екты ОС, как файлы, процессы и потоки. Все эти объекты могут находиться в двух состояниях: сигнальном и несигнальном — свободном. Для каждого объекта смысл, вкладываемый в понятие «сигнальное состояние», зависит от типа объек­та. Так, например, поток переходит в сигнальное состояние тогда, когда он завер­шается. Процесс переходит в сигнальное состояние тогда, когда завершаются все его потоки. Файл переходит в сигнальное состояние в том случае, когда заверша­ется операция ввода-вывода для этого файла. Для остальных объектов сигнальное состояние устанавливается в результате выполнения специальных системных вы­зовов. Приостановка и активизация потоков осуществляются в зависимости от состояния синхронизирующих объектов ОС. Потоки с помощью специального системного вызова сообщают операционной сис­теме о том, что они хотят синхронизировать свое выполнение с состоянием неко­торого объекта. Будем далее называть этот системный вызов Wait(X), где X — ука­затель на объект синхронизации. Системный вызов, с помощью которого поток может перевести объект синхронизации в сигнальное состояние, назовем Set(X). Поток может ожидать установки сигнального состояния не одного объекта, а не­скольких. При этом поток может попросить ОС активизировать его при установ­ке либо одного из указанных объектов, либо всех объектов. Поток может в каче­стве аргумента системного вызова WaitO указать также максимальное время, которое он будет ожидать перехода объекта в сигнальное состояние, после чего ОС должна его активизировать в любом случае. Может случиться, что установки некоторого объекта в сигнальное состояние ожидают сразу несколько потоков. В зависимости от объекта синхронизации в состояние готовности могут перево­диться либо все ожидающие это событие потоки, либо один из них. Синхронизация тесно связана с планированием потоков. Во-первых, любое об­ращение потока с системным вызовом Wait(X) влечет за собой действия в под­системе планирования — этот поток снимается с выполнения и помещается в очередь ожидающих потоков, а из очереди готовых потоков выбирается и акти­визируется новый поток. Во-вторых, при переходе объекта в сигнальное состоя­ние (в результате выполнения некоторого потока — либо системного, либо при­кладного) ожидающий этот объект поток (или потоки) переводится в очередь готовых к выполнению потоков. В обоих случаях осуществляется перепланиро­вание потоков, при этом если в ОС предусмотрены изменяемые приоритеты и/или кванты времени, то они пересчитываются по правилам, принятым в этой опера­ционной системе. Рассмотрим несколько примеров, когда в качестве синхронизирующих объектов используются файлы, потоки и процессы. Пусть программа приложения построена так, что для выполнения запросов, по­ступающих из сети, основной поток создает вспомогательные серверные потоки. При поступлении от пользователя команды завершения приложения основной поток должен дождаться завершения всех серверных потоков и только после этого завершиться сам. Следовательно, процедура завершения должна включать вызов Wait(Xl, Х2, ...), где XI, Х2 — указатели на серверные потоки. В результате выпол­нения данного системного вызова основной поток будет переведен в состояние ожидания и останется в нем до тех пор, пока все серверные потоки не перейдут в сигнальное состояние, то есть завершатся. После этого ОС переведет основной поток в состояние готовности. При получении доступа к процессору основной поток завершится. Другой пример. Пусть выполнение некоторого приложения требует последова­тельных работ-этапов. Для каждого этапа имеется свой отдельный процесс. Сиг­налом для начала работы каждого следующего процесса является завершение предыдущего. Для реализации такой логики работы необходимо в каждом про­цессе, кроме первого, предусмотреть выполнение системного вызова Wait(X), в котором синхронизирующим объектом является предшествующий поток. Объект-файл, переход которого в сигнальное состояние соответствует заверше­нию операции ввода-вывода с этим файлом, используется в тех случаях, когда поток, инициировавший эту операцию, решает дождаться ее завершения, прежде чем продолжить свои вычисления. Однако круг событий, с которыми потоку может потребоваться синхронизиро­вать свое выполнение, отнюдь не исчерпывается завершением потока, процесса или операции ввода-вывода. Поэтому в ОС, как правило, имеются и другие, бо­лее универсальные объекты синхронизации, такие как *событие (event), мьтекс (mutex), системный семафор*и другие. Мьютекс, как и семафор, обычно используется для управления доступом к дан­ным. В отличие от объектов-потоков, объектов-процессов и объектов-файлов, которые при переходе в сигнальное состояние переводят в состояние готовности все по­токи, ожидающие этого события, объект-мьютекс «освобождает» из очереди ожи­дающих только один поток. Работа мьютекса хорошо поясняется в терминах «владения». Пусть поток, кото­рый, пытаясь получить доступ к критическим данным, выполнил системный вы­зов Wait(X), где X — указатель на мьютекс. Предположим, что мьютекс находится в сигнальном состоянии, в этом случае поток тут же становится его владельцем, устанавливая его в несигнальное состояние, и входит в критическую секцию. После того как поток выполнил работу с критическими данными, он «отдает» мьютекс, устанавливая его в сигнальное состояние. В этот момент мьютекс сво­боден и не принадлежит ни одному потоку. Если какой-либо поток ожидает его освобождения, то он становится следующим владельцем этого мьютекса, одно­временно мьютекс переходит в несигнальное состояние. Объект-событие (в данном случае слово «событие» используется в узком смыс­ле, как обозначение конкретного вида объектов синхронизации) обычно исполь­зуется не для доступа к данным, а для того, чтобы оповестить другие потоки о том, что некоторые действия завершены. Пусть, например, в некотором прило­жении работа организована таким образом, что один поток читает данные из файла в буфер памяти, а другие потоки обрабатывают эти данные, затем первый поток считывает новую порцию данных, а другие потоки снова ее обрабатывают и так далее. В начале работы первый поток устанавливает объект-событие в не­сигнальное состояние. Все остальные потоки выполнили вызов Wait(X), где X — указатель события, и находятся в приостановленном состоянии, ожидая наступ­ления этого события. Как только буфер заполняется, первый поток сообщает об этом операционной системе, выполняя вызов Set(X). Операционная система просматривает очередь ожидающих потоков и активизирует все потоки, которые ждут этого события.

*Сигналы*

*Сигнал*дает возможность задаче реагировать на событие, источником которого может быть операционная система или другая задача. Сигналы выбывают пре­рывание задачи и выполнение заранее предусмотренных действий. Сигналы мо­гут вырабатываться синхронно, то есть как результат работы самого процесса, а могут быть направлены процессу другим процессом, то есть вырабатываться асинхронно. Синхронные сигналы чаще всего приходят от системы прерываний процессора и свидетельствуют о действиях процесса, блокируемых аппаратурой, например деление на нуль, ошибка адресации, нарушение защиты памяти и т. д. Примером асинхронного сигнала является сигнал с терминала. Во многих ОС предусматривается оперативное снятие процесса с выполнения. Для этого поль­зователь может нажать некоторую комбинацию клавиш (Ctrl+C, Ctrl+Break), в ре­зультате чего ОС вырабатывает сигнал и направляет его активному процессу. Сигнал может поступить в любой момент выполнения процесса (то есть он явля­ется асинхронным), требуя от процесса немедленного завершения работы. В дан­ном случае реакцией на сигнал является безусловное завершение процесса. В системе может быть определен набор сигналов. Программный код процесса, которому поступил сигнал, может либо проигнорировать его, либо прореагиро­вать на него стандартным действием (например, завершиться), либо выполнить специфические действия, определенные прикладным программистом. В послед­нем случае в программном коде необходимо предусмотреть специальные систем­ные вызовы, с помощью которых операционная система информируется, какую процедуру надо выполнить в ответ на поступление того или иного сигнала. Сигналы обеспечивают логическую связь между процессами, а также между про­цессами и пользователями (терминалами). Поскольку посылка сигнала преду­сматривает знание идентификатора процесса, то взаимодействие посредством сигналов возможно только между родственными процессами, которые могут по­лучить данные об идентификаторах друг друга. В распределенных системах, состоящих из нескольких процессоров, каждый из которых имеет собственную оперативную память, блокирующие переменные, се­мафоры, сигналы и другие аналогичные средства, основанные на разделяемой памяти, оказываются непригодными. В таких системах синхронизация может быть реализована только посредством обмена сообщениями.

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

Процессоры Intel начиная с Pentium Pro позволяют операционным системам применять одно-, двух- и трехступенчатые схемы. И даже разрешается одновременное использование страниц различного размера. Эта возможность, конечно, повысила бы эффективность страничного преобразования, будь она внедрена в Windows NT. Увы, эта ОС возникла раньше и поддерживает только двухступенчатую схему преобразования с фиксированным размером страниц. Размер страниц для платформы Intel составляет 4 Кбайт, а для DEC Alpha - 8 Кбайт. Схема страничного преобразования (рис. 1) выглядит так:

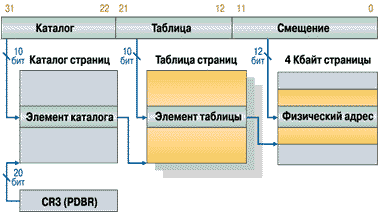


Рисунок 1. Схема страничного преобразования адреса для платформы Intel

32-разрядный линейный адрес разбивается на 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 виртуальной памяти. Размер страницы - 4 Кбайт, и младших 12 разрядов линейного адреса как раз хватает (2^12 = 4096), чтобы определить точный физический номер адресуемой ячейки памяти внутри этой страницы.

Рассмотрим отдельный элемент таблицы страниц (PTE - Page Table Element) более подробно, так как он содержит массу полезной информации (рис 2).

Picture 2.

Старшие пять бит определяют тип страницы с точки зрения допустимых операций. Win32 API поддерживает три допустимых значения этого поля: PAGE\_NOACCESS, PAGE\_READONLY и PAGE\_READWRITE. Следующие 20 бит определяют базовый физический адрес страницы в памяти. Если дополнить их 12 младшими разрядами линейного адреса, они образуют физический адрес ячейки памяти, к которой производится обращение. Следующие четыре бита PTE описывают используемый файл подкачки. Комбинацией этих битов ссылаются на один из 16 возможных в системе файлов. Последние три бита определяют состояние страницы в системе. Старший из них (T-Transition) отмечает страницу как переходную, следующий (D-Dirty) - как страницу, в которую была произведена запись. Информация об изменениях в странице необходима системе для того, чтобы принять решение о сохранении страницы в файле подкачки при ее вытеснении (принудительном освобождении занятой памяти). Действительно, если страница не изменялась в памяти после загрузки, то ее можно просто стереть, ведь в файле подкачки сохранилась ее копия. И наконец, младший бит (P-Present) определяет, присутствует ли страница в оперативной памяти или же она находится в файле подкачки

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

Типы ОС:

*Универсальные ОС*- отличаются тем, что могут решать широкий спектр поставленных задач при удовлетворительном использовании ресурсов вычислительной системы.

Особенности: широкое распространение, динамично развивающийся интерфейс, наличие интегрированных средств разработки прикладных программ, отсутствие жестких требований к эффективности, скорости обработки, надежности хранения и времени реакции системы.

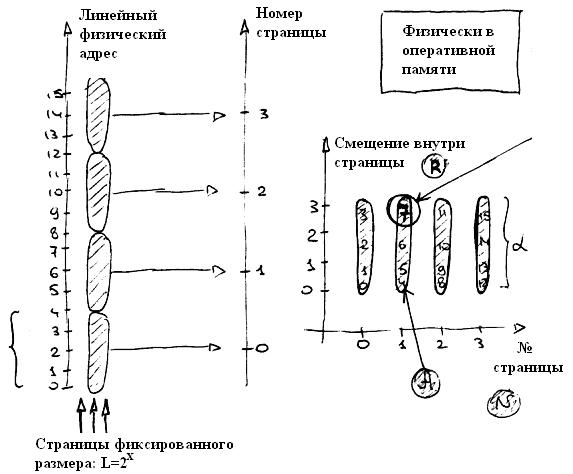
*ОС специального назначения*- для них существует ряд требований, накладывающих жесткие ограничения на параметры работы вычислительной системы. Подразделяются на ОС реального времени, сетевые ОС, защищенные системы.

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

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=k\*L+R для вычисления адреса. Например, от адреса (1,3) мы используем A=1\*2^2+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.

*Механизм преобразования виртуального адреса в физический.*

При обращении к ОП выполняются следующие действия:

1) зная неполный адрес таблицы страниц, зная номер виртуальной страницы и зная длину записи в таблице находят нужную запись

2) считывается номер физической страницы

3) к номеру физической страницы присоединяется смещение, используя операцию конкатенации

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

Основными функциями файловой системы являются:

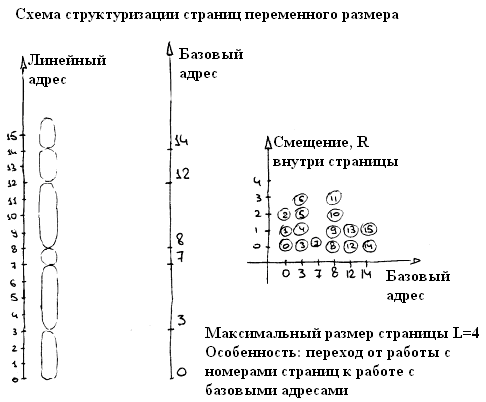
* размещение и упорядочивание на носителе данных в виде файлов;
* определение максимально поддерживаемого объема данных на носителе информации;
* создание, чтение и удаление файлов;
* назначение и изменение атрибутов файлов (размер, время создания и изменения, владелец и создатель файла, доступен только для чтения, скрытый файл, временный файл, архивный, исполняемый, максимальная длина имени файла и т.п.);
* определение структуры файла;
* поиск файлов;
* организация каталогов для логической организации файлов;
* защита файлов при системном сбое;
* защита файлов от несанкционированного доступа и изменения их содержимого.

Функционал файловой системы нацелен на решение следующих задач:

* присвоение имен файлам;
* программный интерфейс работы с файлами для приложений;
* отображение логической модели файловой системы на физическую организацию хранилища данных;
* поддержка устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
* содержание параметров файла, необходимых для правильного взаимодействия с другими объектами системы (ядро, приложения и пр.).

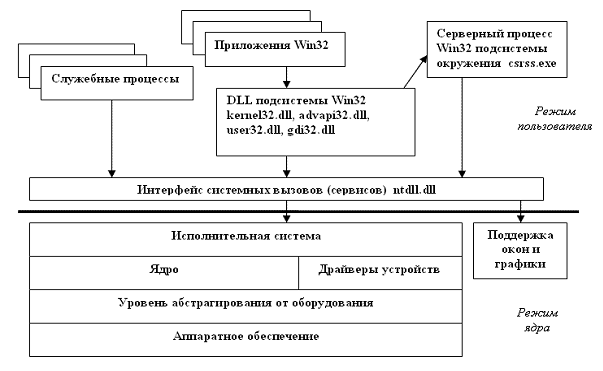
В многопользовательских системах реализуется задача защиты файлов от несанкционированного доступа, обеспечение совместной работы. При открытии файла одним из пользователей для других этот же файл временно будет доступен в режиме «только чтение».

**13. 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.

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



**Рис.**Упрощенная архитектурная схема ОС Windows

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 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 вышележащим 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.

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

Этапы:

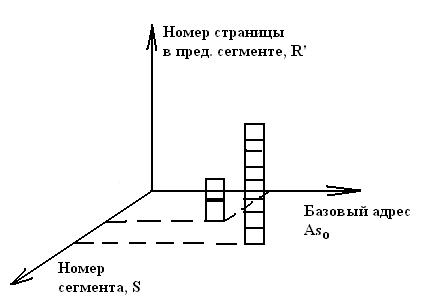
1-Исходное пространство структурируют исходными страницами.

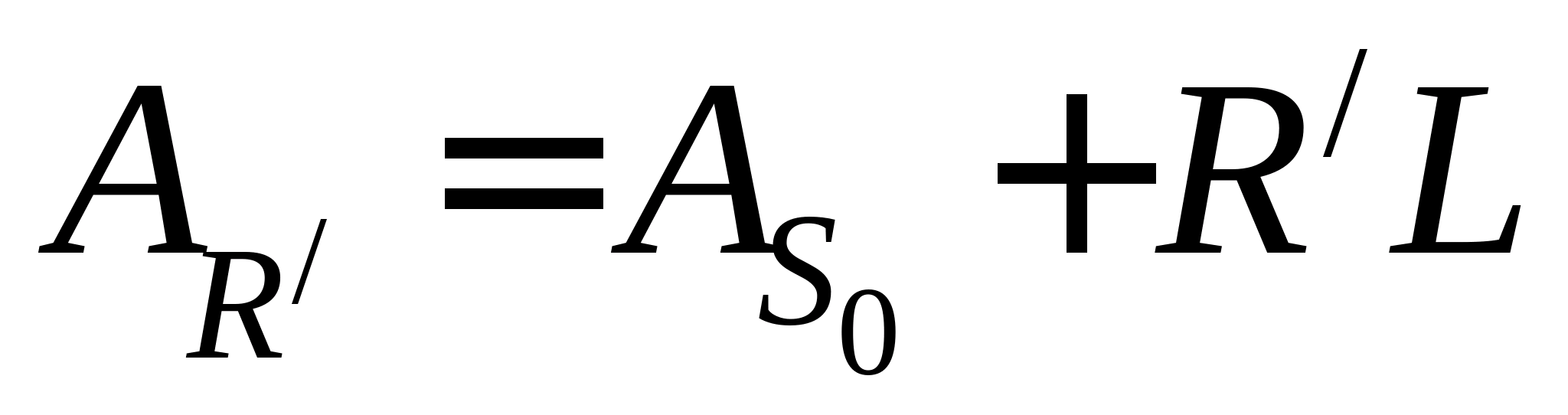
2-Сегмент рассматривается как уже некоторая непрерывная последовательность номеров страниц. Размер сегмента, это количество страниц.

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.

**16. 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.

**17. Опишите виды групп информации дескриптора.**

*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.

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

С каждым файлом связан набор атрибутов (характеристик), то есть набор сведений о файле. Состав атрибутов может сильно различаться для разных файлов системы. Возможны следующие атрибуты:

*Имя файла*. В старых версиях операционных систем длина имени была жестко ограничена 6-8 символами с целью экономии места для хранения имени и ускорения работы. В настоящее время максимальная длина имени составляет 255 символов, что позволяет при желании включить в имя файла подробное описание его содержимого.

*Расширение имени.* Включает в себя правую часть имени, отделяемую точкой. В настоящее время является частью имени. Указывает на тип файла.

*Размер файла.* Принято указывать в байтах, но в настоящее время еще часто указывается в тех единицах, в которых файл занимает наименьший размер (например, 6,29 МБ (6 602 180 байт)).

*Временные штампы.* Под этим термином понимаются различные отметки даты и времени. Как правило, наиболее важным является время последней модификации и время создания файла.

*Номер версии.* В некоторых операционных системах при всяком изменении файла создавалась его новая версия, причем система могла сохранить либо все версии, либо только несколько последних. Этот атрибут не используют из-за большой избыточности. В результате настоящее время используются системы сохранения версий только в области программирования, чтобы всегда была возможность вернуться к рабочей версии программы.

*Владелец файла.* Этот атрибут необходим в многопользовательских операционных системах для организации защиты данных. Как правило, владельцем является пользователь, который создал этот файл. Иногда кроме индивидуального владельца указывается группа пользователей, как коллективный владелец.

*Флаги (битовые атрибуты).* Их разнообразие ограничивается лишь фантазией разработчика системы, но наиболее распространенным и важным является флаг «только для чтения», защищающий файл от случайного изменения или удаления. В зависимости от возможностей конкретной файловой системы, файл может быть отмечен как «сжатый», «шифрованный» и т.п.

*Данные о размещении файла на диске.* Пользователь, как правило, не знает и не хочет знать о размещении файла. Для системы эти данные необходимы чтобы найти файл.

*Тип файла:*

* *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).

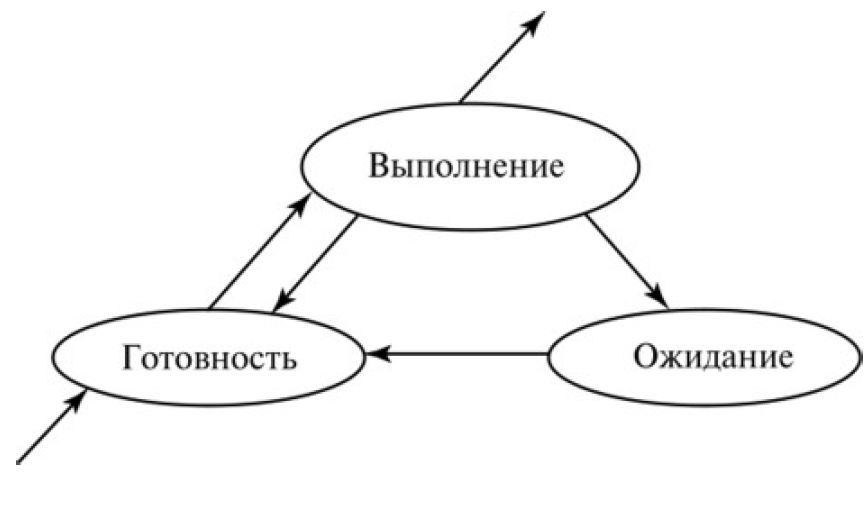
*По другой классификации файлы бывают:*

* 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.

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



*Порождение* – подготавливаются все условия для выполнения.

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

*Выполнение* – 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.

Переход Выполнение-Ожидание представляет собой блокировку процесса, которая может произойти при вызове блокирующей системной функции.

Переход Ожидание-Готовность – это пробуждение процесса, оно выполняется системой при возникновении соответствующего условия.

Переход Выполнение-Готовность ранее не рассматривался. Он называется вытеснением процесса и выполняется системой, когда она принимает решение о смене текущего процесса.

Для обратного перехода Готовность-Выполнение нет общепринятого термина. Будем называть его выбором процесса для выполнения. Отметим, что этот переход почти всегда связан либо с блокировкой, либо с вытеснением прежнего текущего процесса.

**20. Приведите классификацию процессов.**

По временным характеристикам различают *интерактивные*, *пакетные* процессы и процессы *реального времени*. Время существования интерактивного процесса определяется реакцией ЭВМ на запрос обслуживания и составляет секунды. Процессы реального времени имеют гарантированное время окончания работы и время реакции мсек. Пакетные процессы запускаются один вслед за другим и время реакции часы и более.

*По генеалогическому признаку* различают *порождающие* и *порожденные* процессы.

*По результативности* различают *эквивалентные*, *тождественные* и *равные* процессы. Все они имеют одинаковый конечный результат, но эквивалентные процессы могут реализовываться как на одном, так и на многих процессорах по одному или разным алгоритмам, то есть они имеют разные трассы, которые определяют порядок и длительность пребывания процесса в разных состояниях. Тождественные процессы реализуются по одной и той же программе, но имеют разные трассы. Одинаковые процессы реализуются по одной программе и имеют одинаковые трассы. *По времени развития* процессы делятся на *последовательные*, *параллельные* и *комбинированные*. Для последних есть точки, в которых существуют оба процесса, и точки, в которых существует только один процесс.

*По месту развития* процессы делятся на *внутренние* и *внешние*. Внутренние - реализуются на центральном процессоре, внешние- реализуются на внешних процессорах.

*По принадлежности* к операционной системе процессы бывают *системные* (исполняют программу из состава операционной системы) и *пользовательские*.

*По связности* различают процессы:

­ 1) взаимосвязанные, которые имеют какую-то связь (пространственно-временную, управляющую, информационную);

­ 2) изолированные – слабо связанные;

­ 3) информационно-независимые, которые используют совместные ресурсы, но имеют собственные информационные базы;

­ 4) взаимодействующие – имеют информационные связи и разделяют общие структуры данных;

­ 5) взаимосвязанные по ресурсам;

­ 6) конкурирующие.

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

Дисциплиной обслуживания называется *правило* или *набор правил*, по которым выбирается заявка из очереди на обслуживание.

Различают дисциплины обслуживания:

1) без приоритетов;

2) с приоритетами.

*Дисциплины обслуживания без приоритетов*характеризуются преимущественным правом на обслуживание каждой пришедшей заявки, и выборка заявки из очереди осуществляется в соответствии с каким-либо правилом.

Такова дисциплина обслуживания ***FIFO*** (*First In – First Out*, т.е. первым пришел – первым вышел: заявки выбираются из очереди по порядку). Такая дисциплина обслуживания называется также *круговой*.

Примером кругового обслуживания может служить обычная очередь к кассе в магазине или к пункту инструментального контроля автомобилей.

Также бесприоритетной является дисциплина обслуживания ***LIFO*** (*Last In – First Out*, т.е. последним пришел – первым вышел). В соответствии с этой дисциплиной обслуживания действует так называемая *магазинная*, или *стековая* память.

Примером такого обслуживания может служить заряжание патронов в обойму пистолета и их отстреливание.

***RAND*** –обслуживание в произвольном порядке.

*Дисциплины обслуживания с приоритетами.* В данном случае заявки ранжируются в зависимости от приоритета. Должны существовать минимум два уровня приоритета.

Существуют следующие основные типы приоритетов:

1) абсолютный приоритет;

2) относительный приоритет;

3) динамический приоритет.

*Абсолютный приоритет.* Обслуживание более приоритетной заявки начинается сразу после ее поступления, а обслуживание предыдущей заявки прерывается.

Примером может служить междугородный телефонный звонок, который разъединяет предыдущий разговор, что называется «на полуслове».

*Относительный приоритет.*Сначала заканчивается обслуживание текущей заявки и только затем инициируется обслуживание поступившей, более приоритетной, заявки.

Пример – обслуживание «вне очереди» в парикмахерской клиента, обладающего льготой, например, ветерана боевых действий в «горячих точках».

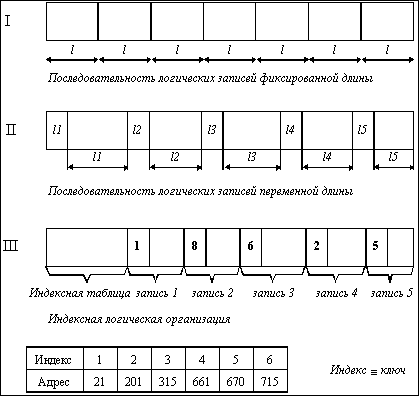
*Динамический приоритет.* Приоритет заявки может изменяться в процессе ее нахождения в системе массового обслуживания.

Существуют и более сложные дисциплины обслуживания.

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

*(хз про то ли это)*

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рисунке 2.33 показаны несколько схем логической организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В файловых системах ОС UNIX и MS-DOS файл имеет простейшую логическую структуру - последовательность однобайтовых записей.



**23. Вытесняющие алгоритмы планирования процессов.**

Вытесняющие (preemptive) алгоритмы — это такие способы планирования потоков, в которых решение о переключении процессора с выполнения одного потока на выполнение другого потока принимается операционной системой, а не активной задачей.

Основным различием между вытесняющими и невытесняющими алгоритмами является степень централизации механизма планирования потоков. При вытесняющем мультипрограммировании функции планирования потоков целиком сосредоточены в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться одновременно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активного потока, запоминает его контекст, выбирает из очереди готовых потоков следующий, запускает новый поток на выполнение, загружая его контекст.

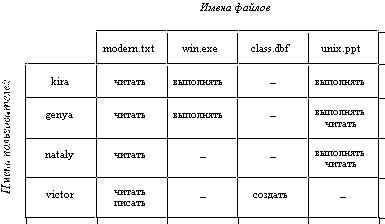
**24. Права доступа к файлу. Основные подходы к определению прав доступа.**

Определить права доступа к файлу – значит, определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

* создание файла,
* уничтожение файла,
* открытие файла,
* закрытие файла,
* чтение файла,
* запись в файл,
* дополнение файла,
* поиск в файле,
* получение атрибутов файла,
* установление новых значений атрибутов,
* переименование,
* выполнение файла,
* чтение каталога,

и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рисунок 2.35). В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.



*Матрица прав доступа*

Различают два основных подхода к определению прав доступа:

* избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
* мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

**25. 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.

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

Сокращение от «Resilient File System», ReFS – эта новая система, созданная на базе NTFS. На данном этапе ReFS не предлагает комплексную замену NTFS для использования на диске домашних пользователей. Файловая система имеет свои преимущества и недостатки.

ReFS предназначена для [*решения основных проблем NTFS*](https://hetmanrecovery.com/ru/recovery_news/recovering-compressed-encrypted-ntfs-files.htm)*.* Она более устойчива к повреждению данных, лучше справляется с повышенной нагрузкой и легко масштабируется для очень больших файловых систем. Давайте рассмотрим, что это означает?

## *ReFS защищает данные от повреждения*

Файловая система использует контрольные суммы для метаданных, а также может использовать контрольные суммы для данных файла. Во время чтения или записи файла, система проверяет контрольную сумму что бы убедиться в её правильности. Таким образом осуществляется обнаружение искаженных данных в режиме реального времени.

*ReFS интегрирована с функцией Дисковое пространство*. Если вы настроили зеркальное хранилище данных, то с помощью ReFS, Windows обнаружит и автоматически устранит повреждение файловой системы, скопировав данные с другого диска. Эта функция доступна как в Windows 10, так и Windows 8.1.

Если файловая система обнаружит поврежденные данные, которые не имеют альтернативной копии для восстановления, то ReFS сразу удалить такие данные с диска. Это не потребует перезагрузки системы или отключения устройства хранения информации, как в случае с NTFS.

Необходимость использования утилиты chkdsk полностью исчезает, так как файловая система автоматически корректируется сразу в момент возникновения ошибки. Новая система устойчива и к другим вариантам повреждения данных. NTFS во время записи метаданных файла записывает их напрямую. Если в это время произойдет отключение питания или сбой компьютера, вы получите повреждение данных.

Во время изменения метаданных ReFS создает новую копию данных и связывает данные с файлом, только после записи метаданных на диск. Это исключает возможность повреждения данных. Эта функция называется копированием на запись, она присутствует и в других популярных ОС Linux системах: ZFS, BtrFS, а также файловой системе Apple APFS.

## *В ReFS удалены некоторые ограничения NTFS*

ReFS более современна и поддерживает гораздо большие объемы и более длинные имена файлов чем NTFS. В долгосрочной перспективе это важные улучшения. В файловой системе NTFS имя файла ограничено 255 символами, в ReFS имя файла может содержать до 32768 символов. Windows 10 позволяет отключить ограничение на предел символов для файловых систем NTFS, но он всегда отключается на томах ReFS.

В ReFS больше не поддерживаются короткие имена файлов в формате DOS 8.3. На томе NTFS вы можете получить доступ к *C:\Program Files\* в *C:\PROGRA~1\* для обеспечения совместимости со старым программным обеспечением.

NTFS имеет теоретический максимальный объем в размере 16 экзабайт, а у ReFS теоретический максимальный объем – 262144 экзабайт. Хотя сейчас это не имеет большого значения, но компьютера постоянно развиваются.

## *Какая файловая система быстрее ReFS или NTFS?*

ReFS разрабатывалась не для повышения производительности файловой системы по сравнению с NTFS. Microsoft сделала систему ReFS намного эффективнее в строго определённых случаях.

Например, при использовании с Дисковым пространством, ReFS поддерживает «оптимизацию в режиме реального времени». Допустим у вас есть пул накопителей с двумя дисками, один обеспечивает максимальную производительность, другой используется для объема. ReFS всегда будет записывать данные на более быстрый диск, обеспечивая максимальную производительность. В фоновом режиме файловая система автоматически переместит большие куски данных на более медленные диски для продолжительного хранения.

В Windows Server 2016 Microsoft улучшила ReFS, для обеспечения лучшей производительности функций виртуальной машины. Виртуальная машина Microsoft Hyper-V использует эти преимущества (теоретически, любая виртуальная машина может использовать преимущества ReFS).

Например, ReFS поддерживает клонирование блоков, это ускоряет процесс клонирования виртуальных машин и операций слияния контрольных точек. Чтобы создать копию виртуальной машины, ReFS нужно только записать новые метаданные на диск и указать ссылку на уже существующие данные. Это связано с тем, что в ReFS несколько файлов могут указывать на одни и те же базовые данные на диске.

Когда виртуальная машина записывает новые данные на диск, они записываются в другое место, а исходные данные виртуальной машины остаются на диске. Это значительно ускоряет процесс клонирования и требует гораздо меньшей пропускной способности диска.

ReFS также предлагает новую функцию «редкого VDL», которая позволяет ReFS быстро записывать нули в большой файл. Это значительно ускоряет создание нового, пустого файла виртуального жесткого диска фиксированного размера (VHD). В NTFS эта операция может занять 10 минут, в ReFS – несколько секунд.

## *Почему ReFS не может заменить NTFS*

Не смотря на ряд преимуществ ReFS не может пока заменить NTFS. Windows не может загрузиться с раздела ReFS и требует NTFS. В ReFS не поддерживаются такие функции NTFS как сжатие данных, шифрование файловой системы, жесткие ссылки, расширенные атрибуты, дедупликация данных и дисковые квоты. Но в отличии от NTFS, ReFS позволяет выполнить полное шифрование диска c помощью BitLocker, включая системные структуры диска.

Windows 10 не позволяет отформатировать раздел в ReFS, эта файловая система доступна только в рамках Дискового пространства. ReFS защищает данные используемые на пулах из нескольких жестких дисков от повреждения. В Windows Server 2016 вы можете форматировать тома с помощью ReFS вместо NTFS. Такой том можно использовать для хранения виртуальных машин, но операционная система по-прежнему может загружаться только с NTFS.

**27. 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.

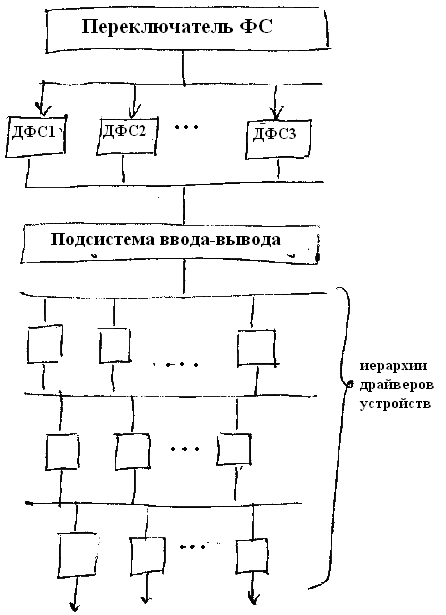
5)-Устойчивость к сбоям. Система должна продолжать функционировать при неисправности отдельного компонента сервера или фрагмента сети. Однако это может приводить к уменьшению производительности или к исключению доступа к некоторой части ФС.

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.

**28. 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.)

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

См. вопрос 8.

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

См. вопрос 8.

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

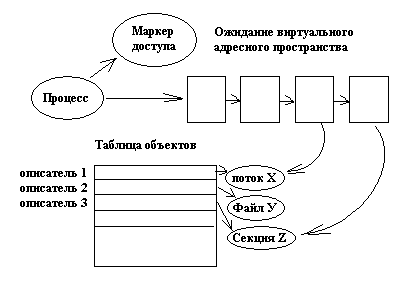
*Процесс* представляет собой один из типов объектов, который связан с другими объектами. К другим можно отнести: поток, секция (совместно используемая память), объект файл, объект порт (для передачи данных м-у процессами), объект маркер доступа (закодированный идентификатор с информацией о праве доступа),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

Если процессу необходимо получить информацию о его маркере доступа, то он должен открыть описатель для своего объекта маркера. Виртуальное адресное пространство создаёт диспетчер. Из ресурса видно, что процесс открыл описателя одного из своих потоков (x), файла и секции совместно используемой памяти. Стрелки от адресного пространства означают виртуальные адреса занятые стеком потока и объектом секциями.

*Атрибуты* – идентификатор процесса, маркер доступа, базовый приоритет, процессорное сродство по умолчанию, размеры квот на ресурсы памяти, время выполнения, счетчик ввода-вывода, счетчик операций виртуальной памяти, коды завершения, порты исключения и отладки.

*Сервисы*– создать процесс, открыть процесс, запросить информацию процесса, установить информацию процесса, выяснить текущий П, завершить П, выделить или освободить виртуальную память, защитить виртуальную память, чтение или запись в виртуальную память, блокировать и разблокировать ВП, опросить ВП и «сбросить» ВП на диск.

**32. 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

**33.Провести сравнительный анализ Android и Windows по свойству надежность.**

Платформа Android является открытой. Мобильное устройство перед использованием требует тщательной настройки, что, с одной стороны, требует времени, а с другой стороны, позволяет пользователю учесть все свои требования к смартфону и настроить его «под себя». Windows занимает место между iOS и Android – настройки минимальны, но всё же существуют. Например, можно изменять размер «плиток» домашнего экрана. Одна из положительных особенностей – это возможность читать и редактировать файлы Word, Excel, PowerPoint и записи OneNote прямо в телефоне. Последние модели смартфонов на Android и Windows предусматривают чтение карт памяти объёмом до 32 Гб.

<https://revolverlab.com/детальное-сравнение-ios-android-и-windows-phone-22f7cc17e6ba>

**34. Провести сравнительный анализ Linux и Windows по свойству защиты.**

Несмотря на то, что ни одна из этих двух ОС не гарантирует абсолютную безопасность, но все же Linux в этом плане заметно превосходит конкурента. Если учесть, что Windows — самая популярная ОС в мире и большинство крупных компаний работают именно на этой платформе, неудивительно, что она же самая интересная для киберпреступников. Linux в этом плане менее привлекательная. Помимо этого, если провести сравнение Linux и Windows, то специфическая архитектура последней делает более простым запуск вредоносных программ — достаточно только дважды кликнуть по файлу типа «.exe». Для поддержания безопасности, пользователям Windows необходимо регулярно обновлять антивирусные программы.

Преимущество Linux в плане безопасности заключает прежде всего в том, что запуск исполняемых бинарных файлов в этой системе более сложный. Кроме того, вирусы в этой системе удалить легче. А если проникнут вредоносные программы, то не затронут системные файлы, принадлежащие root-пользователю. Linux отличается от Windows и тем, что не только не собирает данные о своих пользователях, но также предусматривает возможность встроенного шифрования дисков, что минимизирует риск кражи с устройства конфиденциальной информации.

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

В обычных ОС, обеспечивающих мультипрограммирование, основная память делится на две области, каждая из которых соответствует одному уровню управления. Структура двухуровневой ОС представлена на схеме (рис. 4.6).

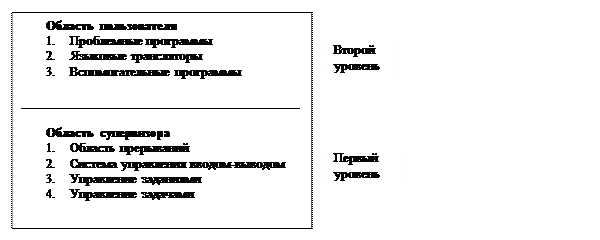


Рис. 4.6. Уровни элементарной операционной системы

Средства первого уровня определяют выполнение программ второго уровня даже в том случае, если в области памяти пользователя находится несколько задач. В системе, использующей виртуальную память, применяется трехуровневая ОС. Резидентный супервизор управляет распределением «реальных» ресурсов и все время находится в ОЗУ. Виртуальная память каждого пользователя делится на две области:

- область привилегированной управляющей программы, обеспечивающей обычное обслуживание;

- область программ пользователя.

Резидентский супервизор и управляющие программы виртуальной памяти существенно отличаются друг от друга.

Pезидентский супервизор:

- в системе присутствует только один резидентский супервизор независимо от количества центральных обрабатывающих устройств;

- работает в состоянии «супервизор»;

- его обработка выполняется без настройки адресов;

- не подвергается временному квантованию.

Управляющие программы виртуальной памяти второго уровня имеют следующие характеристики:

- каждому пользователю приписана одна управляющая программа внутри виртуальной;

- наличие памяти второго уровня, работающей в состоянии «задача»;

- выполняются в режиме настройки адресов (т.е. могут находиться в виртуальной памяти);

- подвергаются временному квантованию;

- могут постранично переноситься в ОЗУ и удаляться из ОЗУ.

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

Все алгоритмы распределения памяти разделены на два класса: алгоритмы, в которых используется перемещение сегментов процессов между оперативной памятью и диском, и алгоритмы, в которых внешняя память не привлекается.

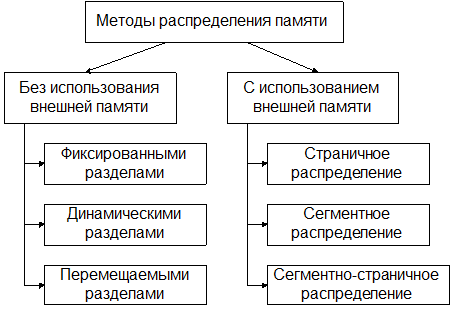


Рисунок 6.3 - Классификация методов распределения памяти

*Распределение памяти фиксированными разделами*

Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько разделов фиксированной величины. Такое разбиение может быть выполнено вручную оператором во время старта системы или во время ее установки. После этого границы разделов не изменяются.

Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь (рисунок 6.4, а), либо в очередь к некоторому разделу (рисунок 6.4, б).

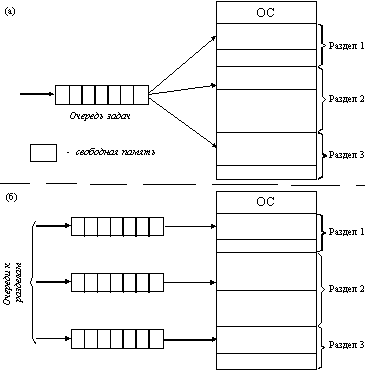


Рисунок 6.4 - Распределение памяти фиксированными разделами:

с общей очередью (а), с отдельными очередями (б)

Подсистема управления памятью в этом случае выполняет следующие задачи:

- сравнивает объем памяти, требуемый для вновь поступившего процесса, с размерами свободных разделов и выбирает подходящий раздел;

- осуществляет загрузку программы в один из разделов и настройку адресов.

При очевидном преимуществе - простоте реализации, данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов. Независимо от размера программы она будет занимать весь раздел. С другой стороны, разбиение памяти на разделы не позволяет выполнять процессы, программы которых не помещаются ни в один из разделов, но для которых было бы достаточно памяти нескольких разделов.

Такой способ управления памятью применялся в ранних мультипрограммных ОС. Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени, в основном благодаря небольшим затратам на реализацию.

*Распределение памяти динамическими разделами*

В этом случае память машины не делится заранее на разделы. Сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память (если достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для него не создается). После завершения процесса память освобождается, и на это место может быть загружен другой процесс. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рисунке 6.5 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так, в момент t0 в памяти находится только ОС, а к моменту t1 память разделена между 5 процессами, причем процесс П4, завершаясь, покидает память. На освободившееся от процесса П4 место загружается процесс П6, поступивший в момент t3.

Функции операционной системы, предназначенные для реализации данного метода управления памятью:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти.

- при создании нового процесса - анализ требований к памяти, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения кодов и данных нового процесса. Выбор раздела может осуществляться по разным правилам, например: «первый попавшийся раздел достаточного размера», «раздел, имеющий наименьший достаточный размер» или «раздел, имеющий наибольший достаточный размер».

- загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей. Данный способ предполагает, что программный код не перемещается во время выполнения, а значит, настройка адресов может быть проведена единовременно во время загрузки.

- после завершения процесса корректировка таблиц свободных и занятых областей.

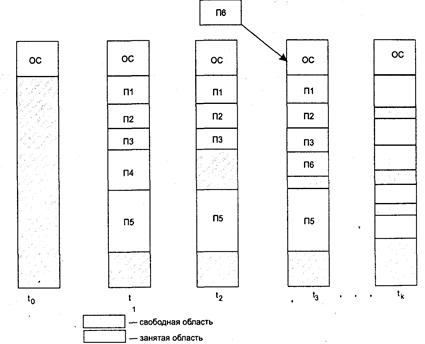


Рисунок 6.5 - Распределение памяти динамическими разделами

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток - *фрагментация памяти*. *Фрагментация* - наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

*Перемещаемые разделы*

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала единую свободную область (рисунок 6.6). В дополнение к функциям, которые выполняет ОС при распределении памяти динамическими разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется *сжатием*.

Сжатие может выполняться либо при каждом завершении процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, а во втором — реже выполняется процедура сжатия.

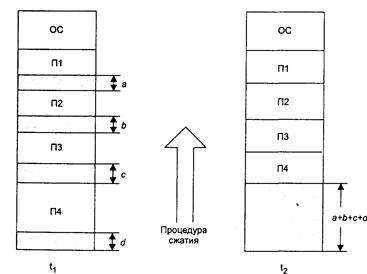


Рисунок 6.6 - Распределение памяти перемещаемыми разделами

Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

**37. Провести сравнительный анализ по основным способам организации вычислительного процесса операционных систем Linux и Windows.**

*Линукс:*

Обычно при загрузке ОС создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, обеспечивающими взаимодействие с пользователями и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции - например, связанные с электронной почтой, Web-страницами, выводом на печать, передачей файлов по сети, периодическим запуском программ (например, дефрагментации дисков) и т.д. Фоновые процессы называют демонами.

Новый процесс может быть создан по запросу текущего процесса. Создание новых процессов полезно в тех случаях, когда выполняемую задачу проще всего сформировать как набор связанных, но, тем не менее, независимых взаимодействующих процессов. В интерактивных системах пользователь может запустить программу, набрав на клавиатуре команду или дважды щелкнув на значке программы. В обоих случаях создается новый процесс и запуск в нем программы. В системах пакетной обработки на мэйнфреймах пользователи посылают задание (возможно, с использованием удаленного доступа), а ОС создает новый процесс и запускает следующее задание из очереди, когда освобождаются необходимые ресурсы. С технической точки зрения во всех перечисленных случаях новый процесс формируется одинаково: текущий процесс выполняет системный запрос на создание нового процесса. Подсистема управления процессами и потоками отвечает за обеспечение процессов необходимыми ресурсами. ОС поддерживает в памяти специальные информационные структуры, в которые записывает, какие ресурсы выделены каждому процессу. Она может назначить процессу ресурсы в единоличное пользование или совместное пользование с другими процессами. Некоторые из ресурсов выделяются процессу при его создании, а некоторые - динамически по запросам во время выполнения. Ресурсы могут быть выделены процессу на все время его жизни или только на определенный период. При выполнении этих функций подсистема управления процессами взаимодействует с другими подсистемами ОС, ответственными за управление ресурсами, такими как подсистема управления памятью, подсистема ввода-вывода, файловая система. Для того чтобы процессы не могли вмешаться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей задачей ОС является изоляция одного процесса от другого. Для этого операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса. В ОС, где существуют процессы и потоки, процесс рассматривается как заявка на потребление всех видов ресурсов, кроме одного - процессорного времени. Этот важнейший ресурс распределяется операционной системой между другими единицами работы - потоками, которые и получили свое название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд. Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации. Работа по определению момента, в который необходимо прервать выполнение текущего потока, и потока, которому следует предоставить возможность выполняться, называется планированием. Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании принимается во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращения к вводу-выводу и другие факторы.

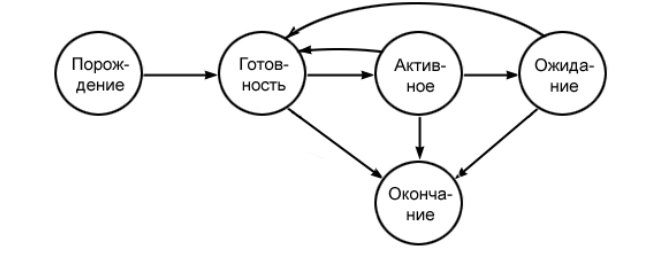
*Windows:*

Обычно процесс создается другим процессом вызовом Win32-функции CreateProcess (а также CreateProcessAsUser и CreateProcessWithLogonW ). *Создание процесса* осуществляется в несколько этапов.

На первом этапе, выполняемом библиотекой *kernel32.dll* в режиме пользователя, на диске отыскивается нужный файл-образ, после чего создается объект "раздел" памяти для его проецирования на адресное пространство нового процесса. На втором этапе выполняется обращение к системному сервису *NtCreateProcess* для создания объекта "процесс". Формируются блоки EPROCESS, KPROCESS и блок переменных окружения *PEB*. Менеджер процессов инициализирует в блоке процесса маркер доступа (копируя аналогичный маркер *родительского процесса*), идентификатор и другие поля. На третьем этапе в уже полностью проинициализированном объекте "процесс" необходимо создать первичный поток. Это, посредством *системного сервиса* *NtCreateThread*, делает библиотека *kernel32.dll*. Затем *kernel32.dll* посылает подсистеме Win32 сообщение, которое содержит информацию, необходимую для выполнения нового процесса. Данные о процессе и потоке помещаются, соответственно, в список процессов и список потоков данного процесса, затем устанавливается *приоритет процесса*, создается структура, используемая той частью подсистемы Win32, которая работает в режиме ядра, и т.д. Наконец, запускается первичный поток, для чего формируются его начальный контекст и стек, и выполняется запуск стартовой процедуры потока режима ядра *KiThreadStartup*. После этого стартовый код из библиотеки C/C++ передает управление функции main() запускаемой программы.

**38. Постройте граф существования процесса для ОС общего назначения.**

В итоге получилось то же что и в 19-ом.



*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

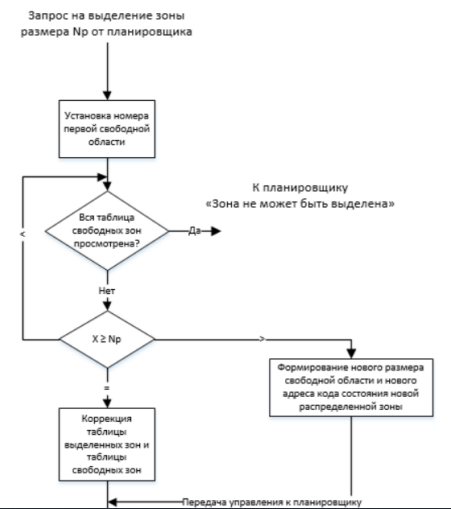
Активное->Готовность окончание времени работы на процессоре, но не завершение процесса

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

**39. 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:

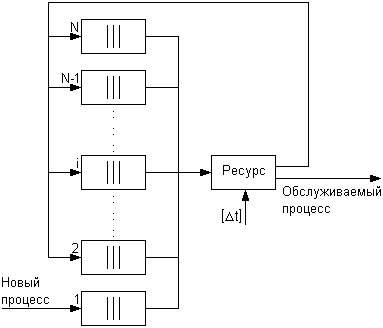
1) Выделение памяти;

2) Учет памяти;

3) Возврат памяти.

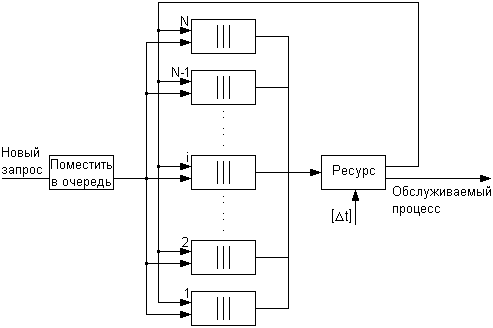
**40. Постройте многоочередную схему управления процессами.**

Простая дисциплина обслуживания:



Организуются N-очередей. Все запросы поступают в конец очереди. Первый процесс из очереди (i) поступает на обслуживание лишь тогда, когда все очереди от (i) до (i – 1) пустые, если кванта времени не хватило, то недо-обслуживаемый процесс поступает в конец очереди с номером (i+1). Если процесс выходит за пределы очереди N, то возможны два варианта: либо он обслуживается до конца, либо по циклическому алгоритму.

Приоритетная много очередная дисциплина обслуживания.



Поступающие процессы попадают в очередь в соответствии с имеющимися приоритетами.

Эти приоритеты определяются параметрами процессов. Во многих ОС алгоритмы планировки построены, как с использованием квантования, так и с использованием приоритетов. Например, в основе планирования может лежать квантование по величине или порядок выбора процесса из очереди определяется приоритетами процессов.

**+ страничная организация виртуальной памяти**

