Ответы на экзаменационные вопросы по ОС.

1. Факторы, оказывающие влияние на развитие ОС.

Огромное влияние на развитие ОС оказывает совершенствование элементной базы и вычислительной аппаратуры, связанных с разработкой новых компьютерных архитектур и таких важнейших устройств как процессоры, различные виды памяти, манипуляторы типа «мышь» и джойстик, жесткие диски, CD-ROM/RW, DVD-ROM/RW, магнитооптические и Blu-ray, а также соответствующие накопители.

Поэтому многие этапы развития ОС тесно связаны с появлением новых типов аппаратных платформ. К ним относятся появившиеся в разное время мини-, микро- и суперкомпьютеры, микропроцессоры, ПК, процессоры и компьютеры с полным и сокращенным набором команд , многопроцессорные и многоядерные архитектуры и другие.

Серьезную эволюцию ОС претерпели и в связи с бурным развитием мультимедиа, а также сетевых технологий.

Важнейшими вехами эволюции ОС в разное время явились следующие средства и механизмы: пакетная обработка, мультипрограммирование, удаленный ввод заданий, прерывания, драйверы внешних устройств (ВУ) или устройств ввода-вывода (УВВ), спулинг, файловые системы, системы реального времени, разделение времени, виртуальная память, свопинг, многозадачность и многопоточность, мультипроцессирование, панели и окна для представления информации, скроллинг, графика и пиктограммы, многооконный графический интерфейс пользователя (ГИП) и многие другие .

На развитие ОС безусловно оказывает постоянное влияние и всевозможное прикладное программное обеспечение (ПО) различного назначения, функционирующее в среде ОС, для управления которым, по сути, ОС и предназначены. Причем от возможностей ОС и качества выполнения ими своих функций во многом зависят успех использования прикладного ПО, возможности, направления и темпы его совершенствования и развития.

Все отмеченные факторы, а также множество других, например, экономических, социальных и даже психологических, являются значимыми ускорителями эволюции ОС.

2. Понятие и определения ОС.

В состав ПО каждого компьютера входит ОС – важнейшая программная система (ПС), с которой обязан уметь работать любой пользователь. Существует много различных определений ОС , рассмотрим некоторые из них.

Определение 1.

ОС – это программа, которая контролирует работу системных и прикладных программ и исполняет роль интерфейса между программным и аппаратным обеспечением компьютера.

Определение 2.

ОС – это часть ПО, которая осуществляет планирование и организацию процесса обработки данных, ввод-вывод, управление данными, распределение ресурсов, подготовку и отладку программ и другие вспомогательные операции.

Определение 3.

ОС – это система программ, которая предназначена для обеспечения определенного уровня эффективности ВС за счет автоматизированного управления ее работой и предоставляемого пользователю определенного набора услуг. ОС включает набор средств проектирования, отладки и выполнения программ, а также управления работой всей ВС.

Определение 4.

ОС – это комплекс программ и данных, которые организуют решение задач и взаимодействие пользователя с техническими средствами САПР.

Определение 5.

ОС – это комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами ВС и прикладными программами, а с другой стороны — предназначены для управления устройствами, управления вычислительными процессами, эффективного распределения вычислительных ресурсов между вычислительными процессами и организации надёжных вычислений .

Определение 6.

ОС – это комплекс системных управляющих и обрабатывающих программ, которые выступают как интерфейс между аппаратурой компьютера и пользователем с его задачами, а также предназначены для наиболее эффективного расходования ресурсов ВС, организации надежных вычислений и удобной работы с ней.

3. Единицы вычислительной работы в среде ОС. Критерии эффективности ВС, подчеркивающие роль ОС.

Задачи, решаемые вычислительной системой(ВС), в среде ОС должны быть оформлены в определенные внутренние единицы вычислительной работы. Обычно они представляют многоуровневую иерархию понятий (сверху вниз):

- Сеанс – ограниченная входом и выходом работа пользователя в среде отдельной ОС при наличии множественной программной среды;
- Задание – последовательный запуск и выполнение нескольких программ, объединенных в пакет. Часть задания – задача;
- Процесс – выполнение на компьютере программы решения некоторой задачи;

- Поток (мини-задача) – асинхронное и независимое (параллельное) выполнение части программы. Поток не существует в системе самостоятельно, он создается и развивается только в рамках процесса.

В ОС Windows NT 4.0 появился еще и облегченный вариант потока – файбер, который отличается тем, что выполняется в контексте создавшего его потока и управлять им должно само приложение.

В наибольшей степени подчеркивают роль ОС следующие критерии эффективности ВС :

- пропускная способность – число задач, выполняемых ВС в единицу времени;
- удобство работы пользователей, связанное, в частности, с возможностью интерактивной работы одновременно с несколькими приложениями на одном компьютере;
- реактивность – способность ВС выдерживать заранее заданные, зачастую очень короткие интервалы времени между запуском программы и получением результата.

Выбранный разработчиками критерий эффективности или их сочетание определяет лицо конкретной ОС, ее архитектуру, организацию, внутренние механизмы, режимы работы и особенности функционирования.

4. Режимы работы компьютеров, обеспечиваемые ОС.

Однопрограммный режим.

В каждый момент времени компьютер используется для решения только одной прикладной задачи по соответствующей программе. При этом часто простаивает процессор во время выполнения операций ввода-вывода или работы пользователя по подготовке следующей программы. Здесь мощная, развитая ОС, в принципе, даже не нужна, так как нет проблем распределения ресурсов.

Однопрограммный пакетный режим.

Данный режим обеспечивает автоматический переход от программы к программе. Его особенности:

- образуется пакет из отдельных программ и данных, после чего формируется задание на его обработку с использованием специализированного формализованного языка управления заданиями (ЯУЗ), содержащее требования к системным ресурсам;
- есть управляющая программа (монитор), которая последовательно запускает программы пакета на решение, вводя исходные данные и выводя результаты или передавая их следующей программе;
- с момента завершения одного пакета до образования следующего процессор компьютера простаивает.

Мультипрограммный пакетный режим.

В данном режиме можно обрабатывать и пакетные задания, формируемые и загружаемые удаленно со своих терминалов всеми пользователями. Так появились первые многотерминальные ОС – системы удаленного (позднее – диалогового) ввода заданий .

Режим реального времени (РВ).

Данный режим характеризуется тем, что

- компьютер средствами ОС осуществляет контроль и управление внешними объектами в темпе поступления данных от каждого объекта управления;
- природа объекта управления может накладывать определенные временные ограничения на обработку таких данных (управление станком или спутником, плавкой стали, стрельбой по нескольким движущимся целям, контроль критического состояния больного). Во всех случаях существует предельно допустимое время, в течение которого должна быть выполнена определенная программа управления объектом, иначе может произойти нежелательное событие.

Режим разделения времени (РДВ).

Данный режим реализуется следующим образом:

- в схему по мультиплексированию (разделению) процессора среди готовых к исполнению программ вводится детерминизм (жесткая определенность, ограничение): каждой программе, готовой к исполнению, планируется для исполнения на процессоре фиксированный, заранее известный интервал времени – **квант** (интервал мультиплексирования), например, $\Delta tm=0,2c$;
- исполняемая программа может не успеть выполниться к моменту окончания кванта. Тогда ее выполнение принудительно прерывается, она переводится в состояние готовности (что равносильно помещению ее в конец очереди готовых к исполнению программ);
- из начала этой очереди извлекается другая программа, получает тот же квант Δtm и т.д.

Многозадачный режим.

Многозадачный режим – это одновременное, параллельное существование и выполнение нескольких задач (заданий, процессов, потоков) с развитыми средствами переключения с одной задачи на другую. Реализованы различные варианты многозадачного режима, основанные на режимах РДВ и мультипрограммном пакетном:

- с вытесняющей многозадачностью (pre-emptive, time-sliced, истинной, независимой), основанной на квантовании времени;
- с кооперативной многозадачностью (cooperative, совместной, не вытесняющей);
- многопоточный режим (multi-threaded, с внутренней вытесняющей многозадачностью).

Для поддержки различных форм и режимов организации вычислительного процесса на компьютерах требуются различные ОС. Каждый тип ОС имеет специфические внутренние механизмы и особые области применения. Некоторые ОС могут поддерживать одновременно несколько режимов, например, одна часть задач может выполняться в режиме пакетной обработки, другая – в режиме РВ или РДВ.

5. Мультипрограммирование. Особенности, достоинства и недостатки мультипрограммного пакетного режима.

Мультипрограммный пакетный режим отличается тем, что

- в ОП находятся несколько равноприоритетных пользовательских программ в виде пакетных заданий, готовых к решению (обслуживанию процессором и ожидающих его освобождения). Из этого пакета заданий формируется **мультипрограммная смесь** – множество одновременно выполняемых задач;
- время работы процессора распределяется между этими программами так, что они образуют очередь к процессору и поочередно решаются на нем;
- при этом каждая активная (выполняющаяся) программа «монополизирует» процессор, освобождая его, только когда он станет ей совсем или временно не нужен;
- параллельно с работой процессора в системе может происходить ввод-вывод – обмен с одним или несколькими УВВ.

Особенности мультипрограммного пакетного режима:

- в данном режиме пропускная способность процессора возрастает, причем с ростом коэффициента мультипрограммирования (числа обрабатываемых программ), так как вероятность простоя процессора падает;
- общее время выполнения задач мультипрограммной смеси по сравнению с предыдущим режимом уменьшается;
- время выполнения отдельного задания может увеличиться на время ожидания освобождения процессора(например, для программы В или С, когда их обмен уже закончен, но процессор пока еще занят). Однако в пакетном режиме это почти незаметно визуально;
- нет гарантии выполнения конкретного задания в течение определенного периода или к определенному моменту времени.

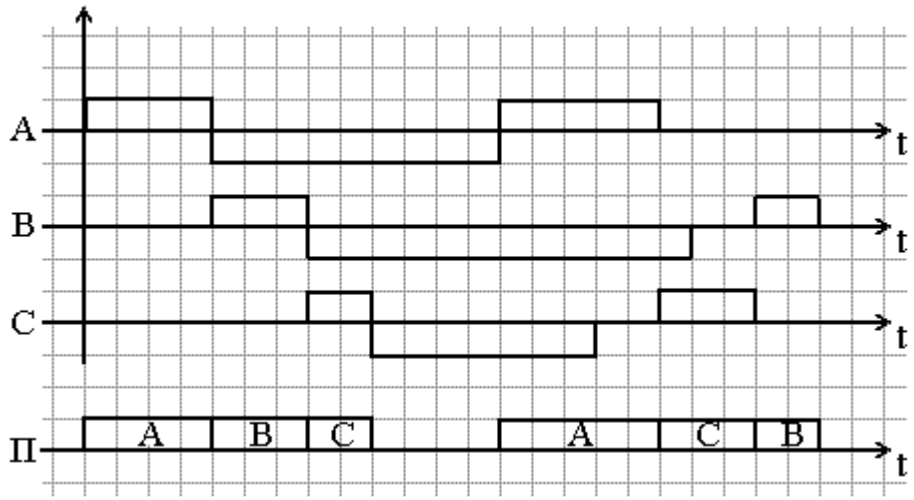
В мультипрограммную смесь лучше включать задания, предъявляющие разные требования к ресурсам и требующие разной интенсивности вычислений и ввода-вывода, чтобы обеспечить сбалансированную загрузку процессора и УВВ.

В данном режиме можно обрабатывать и пакетные задания, формируемые и загружаемые удаленно со своих терминалов всеми пользователями.

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов.

Главной целью и оценкой критерия эффективности таких систем является максимальная пропускная способность.

6. Пример диаграмм работы 3 программ и загрузки процессора для мультипрограммного пакетного режима.



7. Особенности, достоинства и недостатки режима РВ.

Режим **реального времени** (РВ) характеризуется тем, что

- компьютер средствами ОС осуществляет контроль и управление внешними объектами в темпе поступления данных от каждого объекта управления;
- природа объекта управления может накладывать определенные временные ограничения на обработку таких данных (управление станком или спутником, плавкой стали, стрельбой по нескольким движущимся целям, контроль критического состояния больного). Во всех случаях существует предельно допустимое время, в течение которого должна быть выполнена определенная программа управления объектом, иначе может произойти нежелательное событие.

В режиме РВ требуются:

- мгновенный (без задержек) запуск задачи на решение. Значит, место расположения файла задачи должно быть известно заранее, а все необходимые ресурсы выделены ей или хотя бы свободны;
- быстрая (без «чужих» прерываний) отработка важной задачи в соответствии с ее специально установленным высоким приоритетом;
- гарантии того, что задача будет отработана до требуемого срока.

Таким образом, критерием эффективности здесь является способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (и управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство – реактивностью.

Требования ко времени реакции зависят от специфики управляемого объекта: для контроллера робота – менее 1 мс, при моделировании полета – 40 мс.

В системах РВ мультипрограммная смесь представляет фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется по прерываниям от объекта или по плановому расписанию. Способность аппаратуры компьютера и ОС к быстрому ответу зависит в основном от скорости переключения с одной задачи на другую и, в частности, от скорости обработки сигналов прерывания.

Разработчики ОС РВ не стремятся максимально загружать все устройства, они наоборот обычно закладывают существенный запас вычислительной мощности на случай пиковой нагрузки.

8. Особенности, достоинства и недостатки режима РДВ.

Режим **разделения времени** (РДВ, time sharing) реализуется следующим образом:

- в схему по мультиплексированию (разделению) процессора среди готовых к исполнению программ вводится детерминизм (жесткая определенность, ограничение): каждой программе, готовой к исполнению, планируется для исполнения на процессоре фиксированный, заранее известный интервал времени – **квант** (интервал мультиплексирования), например, $\Delta tm=0,2c$;
- исполняемая программа может не успеть выполниться к моменту окончания кванта. Тогда ее выполнение принудительно прерывается, она переводится в состояние готовности (что равносильно помещению ее в конец очереди готовых к исполнению программ);
- из начала этой очереди извлекается другая программа, получает тот же квант Δtm и т.д.

Такое циклическое мультиплексирование процессора, основанное на детерминированной схеме прерываний программ, гарантирует «справедливость» обслуживания пользователей без монополизации процессора.

Если $ti^{раб}$ » Δt_m для i-программы, то она будет получать процессор в среднем через период $\Delta t^{раб}=n*\Delta t_m$, где n – длина очереди.

Это обстоятельство позволило строить на основе метода разделения времени диалоговые многопользовательские режимы работы ЭВМ с помощью дисплеев.

9. Пример диаграмм работы 3 программ и загрузки процессора для режима РДВ.

Рассмотрим варианты выполнения программ A, B и C (см. рис.1.1) в режиме РДВ. Диаграммы работы программ и загрузки процессора для $\Delta t_m = 1$ (в условных единицах) представлены на рис.1.3,а, для $\Delta t_m = 2$ – на рис.1.3,б.

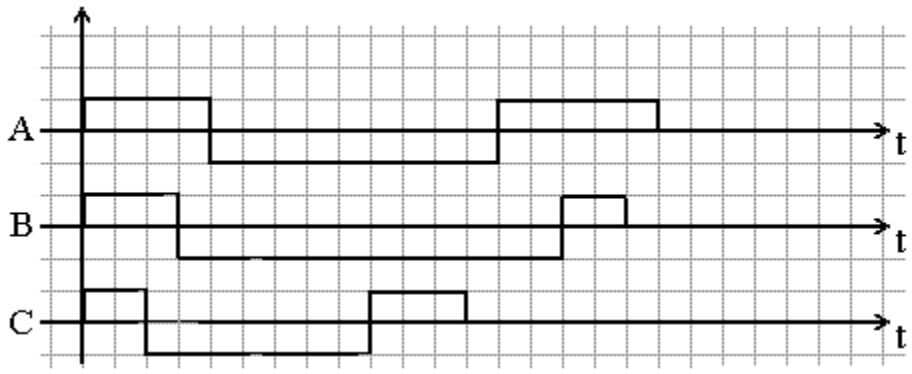


Рис.1.1 Характеристики задач A, B и C

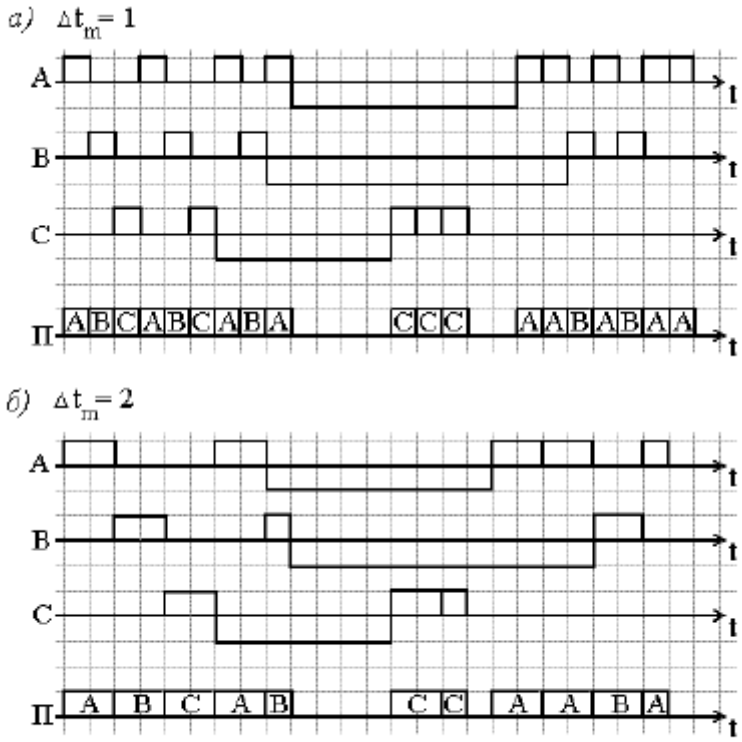


Рис.1.3. Диаграммы работы программ и загрузки процессора в режиме РДВ

В табл.1.1 представлены значения времени решения задач A, B, C и смеси при работе в различных рассмотренных режимах.

Таблица 1.1

Режим	t_A	t_B	t_C	Простой процессора	t_{ABC}
Однопрограммный пакетный	18	17	12	28	47
Мультипрограммный пакетный	18	23	21	4	23
РДВ, Δt_m=1	25	23	16	6	25
РДВ, Δt_m=2	24	23	16	5	24
РДВ, Δt_m=3	26	24	19	7	26
РДВ, Δt_m=4	21	23	20	4	23

Следует отметить, что пропускная способность процессора в режиме РДВ обычно оказывается ниже, чем в мультипрограммном пакетном. Ее снижают более частые переключения процессора с задачи на задачу, а также то, что на выполнение принимается любая задача, а не та, которая выполняется параллельно с работой УВВ (обменом). Поэтому критерием эффективности здесь является удобство и эффективность работы пользователя, а не пропускная способность.

10. Особенности многозадачного режима и варианты его реализации.

Многозадачный режим – это одновременное, параллельное существование и выполнение нескольких задач (заданий, процессов, потоков) с развитыми средствами переключения с одной задачи на другую. Реализованы различные варианты многозадачного режима, основанные на режимах РДВ и мультипрограммном пакетном:

- с вытесняющей многозадачностью (pre-emptive, time-sliced, истинной, независимой), основанной на квантовании времени;
- с кооперативной многозадачностью (cooperative, совместной, не вытесняющей);
- многопоточный режим (multi-threaded, с внутренней вытесняющей многозадачностью).

Таким образом, для поддержки различных форм и режимов организации вычислительного процесса на компьютерах требуются различные ОС. Каждый тип ОС имеет специфические внутренние механизмы и особые области применения. Некоторые ОС могут поддерживать одновременно несколько режимов, например, одна часть задач может выполняться в режиме пакетной обработки, другая – в режиме РВ или РДВ.

11. Функции ОС.

Основные функции ОС:

1. Функция управления оборудованием обеспечивает самый нижний (физический) уровень управления. Основным аппаратом для работы ОС здесь является система прерываний. Используется несколько типов прерываний, для каждого из которых есть своя программа обработки.
Для взаимодействия с УВВ(устройства ввода-вывода) на физическом уровне в ОС предусмотрены специальные программы, которые:

- анализируют состояние УВВ;
- передают данные, если надо преобразуя их;
- обрабатывают сбои УВВ;
- контролируют ошибки.

Для ПК и рабочих станций такие программы называют **драйверами**, для мэйнфреймов и суперкомпьютеров – **канальными программами**. Канальные программы обеспечивают работу каналов: мультиплексного и селекторного, которые сегодня уже практически не применяются.

2. Управление процессами.

3. Управление ресурсами.

4. Управление связями.

Организуется взаимодействие отдельных задач пользователя и программ ОС между собой с передачей данных и управления в соответствии с определенными соглашениями.

5. Управление вводом-выводом.

Организуется работа УВВ на логическом уровне, более удобном и понятном пользователю. Здесь основная единица данных, с которой взаимодействует пользователь – файл, в котором осуществляется поддержка файловых операций.

6. Управление задачами – более высокий по сравнению с процессами уровней управления. Это планирование последовательности выполнения задач. Обычно каждой задаче назначается свой приоритет, в зависимости от которого она чаще или реже получает в свое распоряжение процессор, и ОС должна выбрать из очереди готовых к решению следующую задачу для решения.

7. Управление пользователями (администрирование) – учет пользователей, работающих с системой, защита данных разных пользователей. Некоторые пользователи могут быть привилегированными, число пользователей может меняться.

12.Классификация ОС с примерами известных систем.

Можно выделить 8 следующих признаков классификации ОС.

1. **По назначению ОС**:

- общего назначения (общецелевые), используемые повсеместно;
- специального назначения (специализированные):
    - реального времени;
    - баз данных;
    - мультипроцессорные.
- сетевые, которые могут быть предназначены для поддержки различных видов сетей (по способу управления):
    - равноранговых, когда любая станция может стать и невыделенным (программным) сервером;
    - централизованных (клиент-сервер) – сетей с выделенными (аппаратными) серверами, включающими части.

2. **По режиму обработки задач**:

- однопрограммные;
- многопрограммные:
    - классического мультипрограммирования;
    - разделения времени;
    - многозадачные:
        - с вытесняющей многозадачностью;
        - с кооперативной многозадачностью;
        - многопоточные.

3. **По способу взаимодействия с пользователем**:

- пакетные;
- диалоговые:
    - с диалогом только при подготовке задач к решению;
    - с диалогом на всех этапах обработки задач.

4. **По числу пользователей:** однопользовательские и многопользовательские.

5. **По архитектурным особенностям** компьютера:

- класс компьютера:
  - для мэйнфреймов и суперкомпьютеров;
  - для компьютеров классов «мини» и «супер-мини»;
  - для рабочих станций;
  - для ПК;
- разрядность: 16-разрядные, 32-разрядные, 64-разрядные;
- архитектура процессора:
  - CISC;
  - RISC: POWER PC (IBM), SPARC (SUN Microsystems), MIPS (MIPS), ALPHA (DEC) и другие;
- степень мобильности:
  - «привязанные» к определенной платформе;
  - переносимые;
  - масштабируемые – адаптирующиеся к разным наборам аппаратных ресурсов вычислительного комплекса.

6. По архитектуре ядра: макроядерные (монолитные), микроядерные и гибридные (смешанные).

7. По роли в организации среды:

- основная ОС – устанавливается на жесткий диск первой и часто является единственной;
- дополнительные (гостевые) ОС – устанавливаются в отдельных разделах диска по желанию пользователя после основной, создают собственные множественные прикладные программные среды.

8. По способу работы:

- традиционные ОС – устанавливаются на жесткий диск или другой носитель информации (flash-накопитель);
- Live-ОС – установки не требуют, работают с CD/DVD;
- Online-ОС – устанавливаются и работают в сетевом «облаке» Интернет, как и обычные сетевые (серверные) приложения ОС, вызывающие наибольший интерес сегодня: семейства MS Windows W7/W8, MacOS X, ветвь Linux семейства ОС UNIX .

13.Принципы построения ОС.

1. Частотный принцип.

В алгоритмах и массивах ОС выделяют действия и данные по частоте использования. Для частых действий обеспечиваются условия их быстрого выполнения:

- программы частых действий постоянно находятся в ОП и активно поддерживаются специальными средствами;

- частые операции стараются делать более короткими;
- к часто используемым данным обеспечивается наиболее быстрый доступ.

2. Принцип модульности.

Данный принцип отражает развитие технологии разработки ПО «снизу-вверх». Части ОС обосабливаются в отдельные модули. Модуль – функциональный элемент ОС, имеющий законченное оформление и унифицированные средства сопряжения с другими. Модули затем объединяются в обобщенные модули, группы обобщенных модулей так же – в один и так далее. Легко заменять данный модуль новым.

3. Принцип функциональной избирательности

Принцип функциональной избирательности – логическое продолжение рассмотренных принципов. В ОС выделяется некоторая часть важных модулей, которые должны быть всегда «под рукой» для эффективной организации вычислительного процесса. Эта основа ОС называется ядром.

Как правило, в состав ядра ОС входят: модули управления системой прерываний, средства управления процессами, средства распределения важнейших ресурсов.

4. Принцип генерации.

ОС создается и представляется так, чтобы можно было настраивать (генерировать) ее, исходя из конкретной конфигурации компьютера и круга решаемых проблем. Эта процедура проводится достаточно редко и на долгий период эксплуатации. Процесс генерации реализует программа-генератор (setup, install), она может использовать специальный язык или панели ввода для описания требуемых возможностей, конфигурации компьютера, необходимых компонент ОС. Для генерации используется дистрибутивный комплект (Distribution Kit) на CD, DVD или каких-то других носителях, в результате на жестком диске устанавливается сгенерированная (рабочая) версия ОС.

5. Принцип функциональной избыточности.

Данный принцип предполагает наличие в составе ОС альтернативных однотипных компонент (мониторов, драйверов, загрузчиков и т.д.).

6. Принцип умолчания.

Данный принцип используется для облегчения организации связей с системой при генерации и при последующей работе ОС. ОС хранит базовые описания процессов, модулей, ресурсов, условий выполнения программ и т.п. Значения этих данных считаются заданными по умолчанию, если администратор или пользователь забыл указать или сознательно не указал их. В любом случае ОС сама установит известные ей значения либо воспримет вводимые пользователем.

7. Принцип перемещаемости.

Исполнение модуля не должно зависеть от места его расположения в ОП. Для этого в модулях используются относительные адреса операндов, а ОС должна иметь средства настройки текста модуля по месту его выполнения. Настройка может производиться перед выполнением или во время выполнения, и заключается в определении фактических адресов операндов в ОП для всех команд модуля. Настройка зависит от используемого способа адресации и алгоритма распределения ОП для данной ОС.

8. Принцип защиты.

Данный принцип определяет необходимость разработки мер, ограждающих программы и данные пользователей от искажений или нежелательных влияний друг на друга, пользователей на ОС и наоборот. Программы должны быть гарантированно защищены как при своем выполнении, так и при хранении. Способов влияния чрезвычайно много. Не все их можно предотвратить чисто техническими средствами. Особенно трудно обеспечить защиту, когда используется разделение ресурсов.

9. Принцип независимости программ от внешних устройств позволяет одинаково управлять УВВ независимо от их конкретных физических характеристик.

10. Принцип открытости (прозрачности).

Открытая ОС доступна для анализа пользователями и специалистами. Тем больше приложений можно разработать для такой открытой и понятной всем среды.

11. Принцип наращиваемости (развития).

Наращиваемая (модифицируемая, развиваемая) ОС позволяет не только использовать возможности генерации, но и вводить в ее состав новые модули, совершенствовать существующие, расширяя спектр ее функциональных возможностей.

14.Требования, предъявляемые к ОС.

Главное требование – выполнение своих основных функций по эффективному управлению процессами и ресурсами и обеспечение удобного интерфейса для всех категорий ее пользователей и приложений. Современная ОС, как правило, должна поддерживать мультипрограммную обработку, виртуальную память, свопинг, многооконный ГИП, а также выполнять многие другие необходимые функции и услуги.

Кроме этих требований функциональной полноты ОС предъявляются следующие не менее важные эксплуатационные требования .

1. Производительность.

ОС должна обладать настолько высоким быстродействием и временем реакции, насколько это позволяет используемая аппаратная платформа (Intel, Oracle-SUN, Hewlett Packard, Macintosh). На производительность ОС влияют ее

архитектура, многообразие функций, качество программного кода ОС, сбалансированность аппаратного комплекса компьютера (процессор – (кэш + ОП) – жесткий диск), возможность исполнения ОС на высокопроизводительной (многопроцессорной) платформе и другие.

2. Надежность и отказоустойчивость.

ОС должна быть надежно защищена как от внутренних, так и внешних ошибок, различных сбоев и отказов. Ее действия всегда должны быть предсказуемыми, а приложения не
должны иметь возможности наносить вред ОС. Надежность и отказоустойчивость ОС, прежде всего, определяются архитектурными решениями, положенными в ее основу, а также качеством ее реализации (отлаженностью кода). Кроме того, важно знать, обеспечивает ли ОС программную поддержку аппаратных средств обеспечения отказоустойчивости, таких, например, как дисковые массивы RAID, источники бесперебойного питания UPS, системные средства обеспечения отказоустойчивости (System Fault Tolerance, SFT).

3. Безопасность.

Современная ОС (особенно сетевая) должна защищать данные и другие ресурсы ВС от несанкционированного доступа. Чтобы ОС обладала свойством безопасности, в ее среде, как минимум, должны обеспечиваться: аутентификация (определение легальности пользователей); авторизация (предоставление легальным пользователям дифференцированных прав доступа к ресурсам); аудит (фиксация всех «подозрительных» для безопасности системы событий).

4. Совместимость.

Существует несколько «долгоживущих» популярных ОС (например, семейства UNIX и Linux, MS Windows 2000/XP/Vista/W7/W8, для которых разработана широкая номенклатура приложений (офисных и других). Поэтому пользователю, по какой-либо причине переходящему с одной ОС на другую, необходима возможность запуска своих любимых приложений в среде новой для него ОС. И если данная ОС имеет средства выполнения приложений, написанных для других ОС, то считается, что она обладает совместимостью с этими ОС. Следует различать совместимость на уровне двоичных кодов и совместимость на уровне исходных текстов. Понятие совместимости включает и поддержку ГИП других ОС .

5. Расширяемость.

Несмотря на то, что аппаратура компьютера устаревает очень быстро, полезная жизнь ОС, как в случае с UNIX, может быть долгой. Поэтому лучшие ОС обычно эволюционируют, и эти изменения более значимы и долговременны, чем быстротечные изменения аппаратуры. Эволюция ОС обычно заключается в приобретении ими новых свойств, например, поддержке новых типов УВВ, новых программных, информационных или сетевых технологий. И если код ОС написан

так, что дополнения и изменения могут вноситься без нарушения целостности системы, то такую ОС называют расширяемой. Расширяемость достигается за счет модульной структуры ОС, когда программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс.

6. Переносимость (многоплатформенность).

В идеале код ОС должен легко переноситься на разные типы процессоров и разные аппаратные платформы (отличающиеся не только типом процессора, но и способом организации всей аппаратуры компьютера). Переносимые ОС имеют несколько вариантов реализации для разных аппаратных платформ.

15.Эволюция популярных ОС и их семейств.

Windows 1.0 — 20 ноября 1985 года

В начале 80-х был лишь один компьютер, с которым рядовой пользователь мог общаться на «ты» — Lisa от Apple. Проект оказался провальным, но, с одной стороны, он подготовил почву для Mac, а с другой — создал прецедент использования графического пользовательского интерфейса.

Windows 2.0 — 9 декабря 1987

Microsoft выпускает вторую версию Windows с улучшенной графикой. Windows 2.0 взяла все, что мог дать новый процессор Intel 286, а позже и Intel 386. Был увеличен объем памяти, внедрена функция наложения окон друг на друга, на рабочем столе появились значки, а пользователь мог общаться с системой при помощи «горячих» комбинаций клавиш.

Windows 3.0 — 22 мая 1990 года

С Windows 3.0 начинается успех системы. Пользовательский интерфейс был переделан под 16-цветную гамму, улучшилась работа с памятью.

Windows NT 3.1 — 27 июля 1993 года

Эта 32-битная операционная система была нацелена на бизнес-сегмент и включала многозадачный планировщик для Windows-приложений, интегрированные сети, безопасность сервера домена, OS/2 и POSIX подсистемы, поддержку нескольких процессорных архитектур. В ней же впервые увидела свет файловая система NTFS.

Windows 95 — 24 августа 1995 года

Именно в Windows 95 сформировался привычный всем графический интерфейс с такими элементами, как кнопка «Пуск» со своим меню, панель задач и рабочий стол со значками.

Windows 98 — 25 июня 1998 года

Windows 2000 Professional — 17 февраля 2000 года

Windows Millenium Edition (Me) — 14 сентября 2000 года

Windows XP — 25 октября 2001 года

И вот случилось то, чего все так ждали. Появилась красивая, понятная, быстрая, стабильная Windows, которая сразу же сожгла все мосты.

Windows Vista — 30 января 2007 года

Windows 7 — 22 октября 2009 года

Windows 7 научилась правильно работать с различными сетями, самостоятельно устанавливать драйвера к подключаемым устройствам, получила такую совершенную систему безопасности, которая снимает необходимость устанавливать сторонний антивирус. Также в Windows 7 появилась поддержка сенсорных экранов, которая полностью реализовалась лишь в следующей версии.

Windows 8 — 26 октября 2012 года

Операционная система Windows 8 получила полностью переработанный «плиточный» интерфейс Metro, задачей которого было привести мобильные устройства с сенсорным экраном и компьютеры к схожему пользовательскому опыту. Для удобства работы с ПК без сенсорного экрана в систему был встроен и «классический» рабочий стол. Кнопку «Пуск» убрали, заменив ее «активным углом», нажатие на который открывало стартовый экран с «плитками». Плитки на стартовом экране можно перемещать и группировать, давать группам имена и изменять размер плиток.

Windows 10 — 30 сентября 2014 года

Компания Microsoft была вынуждена признать, что ее попытки сделать интерфейс Metro основным пришлись не по вкусу консервативным пользователям ПК, и ей не оставалось ничего, кроме как вернуть кнопку «Пуск». Также система получит несколько рабочих столов, возможность запуска Metro-приложений в оконном режиме, а также центр уведомлений.

16.Тенденции развития ОС на современном этапе.

1. Развитие ОС происходит непрерывно по стадиям, этапам и фазам в зависимости от значимости достижений, новых технологий и разработок в аппаратном обеспечении компьютеров, периферийных устройств и коммуникаций.

2. Число конкурирующих ОС уменьшилось, выявились лидеры.

3. Свойства конкурирующих ОС сближаются.

4. Выделен ряд свойств, которыми должны обладать все системы.

5. Темпы разработки новых версий ОС увеличиваются, сокращая интервал выпуска новых релизов (с новыми номерами) до 1 – 2 лет.

6. Новая версия ОС, появляющаяся раньше новых версий конкурентов, может раньше захватить рынок, но обычно собирает всю критику конкурентов, друзей и врагов.

7. Новая версия ОС, выходящая последней, не только может учесть все ошибки и просчеты конкурентов, но и обязана сделать это. С другой стороны, чем позже выйдет ОС W7, тем большую долю рынка сумеют захватить MacOS X и Linux, которые выпускают свои дистрибутивы каждые полгода-год, то есть тем будет лучше для конкурентов Microsoft.

8. Успех развития ОС и, в частности, маркетинга, распространения и популярности ОС зависит от дальновидности подходов, правильности и долговременности предлагаемых идей, технических и, в первую очередь, архитектурных решений.

9. Жесткая конкуренция компаний – производителей ОС, в конечном счете, является весомым ускорителем процесса их развития и совершенствования.

10. При наметившихся темпах развития ОС всегда следует ожидать резкого скачка в развитии аппаратуры компьютеров, так как вскоре после появления снижается цена и возрастает степень доступности всех новых высокопроизводительных процессоров, различных акселераторов, Гбайт ОП и сотен Гбайт внешней памяти ПК.

17.Характеристика и определения ресурса.

Смысловое содержание понятия «ресурс» неоднозначно . Известны два его значения:

- *Технический ресурс* – показатель надежности объекта; продолжительность использования объекта или объем работы, выполненной объектом до момента достижения некоторого предельного состояния, когда его использование становится невозможным из-за отказа или нежелательным из-за потери свойств;

- *Материальный ресурс* – сам объект, точнее запас искомых характеристик в составе объекта.

Понятие ресурса в контексте ОС подразумевает сам объект (запас), а не показатель надежности объекта.

Ресурс характеризуют *два свойства*:

- полезность, если есть потребители, которым он необходим. Например, процесс приостанавливается при отсутствии какого-либо необходимого ресурса;
- исчерпаемость, если ресурс может иссякнуть.

Бывают ресурсы исчерпаемые, но воспроизводимые.

Воспроизводимость здесь играет роль противовеса исчерпаемости. Кроме того, исчерпаемый ресурс может со временем просто освободиться вновь.

Границы действия понятия «ресурс» достаточно условны, на чем основано следующее определение.

Определение 1.

Всякий потребляемый, полезный для потребителя объект (независимо от формы его существования) является *ресурсом*.

Из анализа свойств ресурса следует, что

- ресурсы бывают исчерпаемые и неисчерпаемые;
- исчерпаемость ресурсов может вызывать конфликты среди потребителей ресурсов;
- значит, ресурсы надо распределять между потребителями по правилам, устраивающим всех.

Поэтому распределение ресурсов – важнейшая функция ОС.

Можно уточнить смысл ресурса применительно к компьютеру, который содержит функциональные элементы: процессор, каналы, ОП, внешнюю память, принтеры. Все это ресурсы, ценные для развития процессов. Степень детализации ресурса может варьироваться от целого компьютера до 1 разряда его машинного слова.

Процессы, развивающиеся в ВС, используют все ее аппаратные элементы, то есть являются их потребителями. Выделяемые по запросам от процессов элементы ВС являются ее ресурсами.

Определение 2.

Ресурс – средство ВС, которое может быть выделено процессу на определенный интервал времени.

К числу основных ресурсов современных ВС могут быть отнесены: процессоры, основная память, таймеры, наборы данных, дисковые, ленточные и иные накопители, принтеры, сетевые устройства и другие.

18.Понятие и определения процесса.

В литературе по ОС понятие «процесс» является базовым и одновременно наименее точно определенным . Это вид абстракции, которую по-разному истолковывают и используют разные категории лиц. В частности, точки зрения на процесс системных и прикладных программистов расходятся в деталях, формах восприятия и реализации этого понятия. Попробуем определить процесс через понятие «процессор».

Процессор – любое устройство в составе компьютера, способное автоматически выполнять допустимые действия по программе, хранимой в памяти и доступной такому устройству. Тогда помимо центрального процессора можно условно назвать процессором канал ввода-вывода (его и называют «процессором ввода-вывода») или иное устройство, работающее с УВВ. Между процессорами в системе существуют информационные и управляющие связи. Процессор может быть нужен одновременно нескольким пользователям для программы каждого из них. Таким образом, для каждого пользователя, которому нужен процессор, и системы, которая распределяет его между пользователями, и вводится понятие процесса, как некоторой деятельности, связанной с исполнением программы на процессоре.

Определение процесса в контексте ВС (ГОСТ 19781-83). Процесс – это система действий, реализующая определенную функцию в ВС и оформленная так, что управляющая программа ВС может перераспределять ее ресурсы в целях обеспечения мультипрограммирования.

Процесс может протекать по-разному, при этом развитием процесса нужно управлять: дать или изъять процессор у процесса; дать ему результаты работы других процессоров и процессов; дать ему другие ресурсы. А значит, управление процессами – действительно важная функция ОС.

19.Состояния процесса, граф существования, интервал существования, трасса процесса.

При исполнении программы на процессоре различают 5 характерных «активных» состояний процесса (с конкуренцией за ресурсы) :

1) Порождение, когда готовятся условия для первого исполнения программы на процессоре;

2) Активное (счет, выполнение), когда программа исполняется на процессоре;

3) Ожидание (блокирование), когда программа не исполняется на процессоре по причине занятости какого-либо требуемого ресурса, кроме процессора;

4) Готовность, когда программа не исполняется, но для ее исполнения предоставлены все необходимые в данный момент ресурсы, кроме процессора;

5) Окончание – нормальное или аварийное завершение исполнения программы, после чего процессор и другие ресурсы ей больше не предоставляются.

Можно представить состояния и более детально: процесс может находиться в каждом из своих допустимых состояний в течение некоторого интервала времени,

после чего переходит в новое допустимое состояние. Состав допустимых состояний и переходов задают графом существования процесса (ГСП) (рис.2.1).

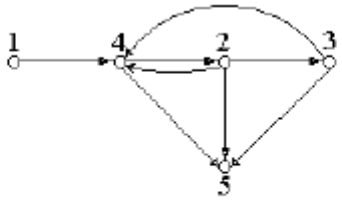


Рис.2.1. Граф существования процесса

Выделяют две временные характеристики, определяющие основные виды, классификационные признаки и свойства процессов [24]:

- **интервал существования процесса** (ИСП) – временной интервал от порождения до окончания процесса;
- **трасса процесса** – порядок переходов на ГСП с учетом длительности пребывания в каждом состоянии.

Определение процесса в контексте ОС.

Для ОС процесс рассматривается как объект, в отношении которого требуется обеспечить реализацию каждого из допустимых состояний, а также допустимые переходы из состояния в состояние в ответ на события, являющиеся причинами этих переходов. Процессы сами могут инициировать такие события, например, затребовать процессор или другой ресурс для исполнения программы.

Часто процесс (задача) кратко определяется как программа в стадии выполнения. Но если программа – статический объект, представляющий файл с кодами и данными, то процесс – динамический объект ОС, возникающий в системе после того, как пользователь или ОС решает запустить программу на выполнение, то есть создать новую единицу вычислительной работы.

20.Классификация процессов.

- Процессы РВ – они должны быть окончены до наступления определенного момента времени;
- Интерактивные – их ИСП составляет 1–3 с, что соизмеримо с реактивностью человека (временем допустимой реакции компьютера на запрос пользователя);
- Пакетные – все остальные.

Пакетные процессы характеризуются 6 классификационными признаками.

1. **По генеалогии** (родословию).

   В любой ОС существующий процесс, который может требовать порождения других процессов. Такой процесс называют порождающим, а процесс, создаваемый по требованию – порождаемым или порожденным. Порожденный процесс в свое время на ИСП может стать и порождающим.

2. **По результативности.**

Здесь ключевые признаки: результат и путь его получения. Важно иметь возможность: воспроизводить результаты процессов, а также сравнивать процессы по их трассам. При этом сравниваться могут: программы (одна или разные), данные (одинаковые), процессоры (один или разные), трассы (одна или разные), результат (одинаковый). С целью сравнения и выделяют 4 вида процессов:

- эквивалентные – имеют одинаковый конечный результат обработки одних и тех же исходных данных по одной или разным программам на одном или разных процессорах. Трассы в общем случае не совпадают;
- тождественные – эквивалентные, одна программа, но трассы не совпадают;
- равные – тождественные и трассы совпадают;
- различные – все остальные.

3. **По временным соотношениям ИСП** (динамическому признаку) выделяют процессы:

- последовательные – их ИСП не пересекаются во времени;
- параллельные – они существуют одновременно на рассматриваемом интервале времени Δt_1 ;
- комбинированные – если имеются два процесса, и на рассматриваемом интервале времени Δt_2 есть хоть одна точка **a**, когда они оба существуют одновременно, и точка **b**, когда существует только один процесс .

4. По отношению к процессору различают процессы внутренние (развиваются на нем) и внешние (развиваются вне него).

5. По отношению к ОС различают процессы системные – при их развитии исполняется программа ОС, и пользовательские – исполняется прикладная программа.

6. По связности выделяют процессы:

- взаимосвязанные – между ними поддерживаются с помощью системы управления процессами связи какого-либо рода (функциональные, временные, управляющие, информационные);
- изолированные – связей нет или есть «слабые», неявные связи;
- информационно-независимые – два взаимосвязанных процесса при развитии используют совместно некоторые ресурсы, но информацией не обмениваются. Их связь может быть функциональная или временная;
- взаимодействующие, когда есть информационные связи, но их схемы и механизмы могут различаться в зависимости от
    - временных соотношений, проявляясь как последовательные, параллельные, комбинированные;

- способа связи, когда он может быть явным – в виде обмена сообщениями, неявным – с помощью разделяемых структур данных;

- конкурирующие – когда есть связь и конкуренция по ресурсам.

21.Отношения между взаимосвязанными процессами.

Если процессы взаимосвязаны, появляются определенные ограничения на порядок их выполнения. Эти ограничения определяют некоторые виды отношений, которые допускаются между процессами. Реализация этих отношений потребует формулировки конкретных синхронизирующих правил.

выделяют основные отношения:

- *Предшествования* – процесс А должен переходить в активное состояние раньше процесса В;
- *Приоритетности* – процесс, имеющий приоритет Р1, может быть переведен в активное состояние при соблюдении двух условий:

  - В состоянии готовности к рассматриваемому процессору нет процессов с приоритетом Р2>Р1;

  - Процессор либо свободен, либо используется процессом с приоритетом Р3<Р1;
- *Взаимного исключения* – два процесса используют обобщенный ресурс (например, память). Так образуется критическая область (КО) – часть ИСП, отражающая действия над этим обобщенным ресурсом в составе одного процесса. КО одного процесса не должна выполняться одновременно с КО над этим же ресурсом в составе другого процесса.

Формулировку конкретных синхронизирующих правил могут осложнять *3 фактора*:

- Динамика развития взаимосвязанных процессов (неопределенность и непредсказуемость порядка и частоты переходов процессов в различные состояния по мере их развития). Все зависит от конкретных ситуаций и условий;
- Для каждой «связки» процессов (в простейшем случае – пары) возникает своя задача синхронизации;
- Разных «связок» процессов может быть много. При этом каждый процесс может участвовать в различном числе «связок».

Могут возникать и другие, дополнительные отношения между процессами, определяющие очевидный порядок их развития типа «сначала-потом»: «производитель-потребитель», «писатель-читатель».

22.Факторы, осложняющие формулировку синхронизирующих правил для взаимосвязанных процессов.

Формулировку конкретных синхронизирующих правил могут осложнять *3 фактора*:

- Динамика развития взаимосвязанных процессов (неопределенность и непредсказуемость порядка и частоты переходов процессов в различные состояния по мере их развития). Все зависит от конкретных ситуаций и условий;

- Для каждой «связки» процессов (в простейшем случае – пары) возникает своя задача синхронизации;
- Разных «связок» процессов может быть много. При этом каждый процесс может участвовать в различном числе «связок».

Могут возникать и другие, дополнительные отношения между процессами, определяющие очевидный порядок их развития типа «сначала-потом»: «производитель-потребитель», «писатель-читатель».

23.Поток: понятие, особенности, роль в среде ОС.

Процесс может требовать для своего выполнения одной или нескольких более мелких работ – **потоков**. Рассмотрим принципиальные отличия потока от процесса.

Любая работа ВС связана с выполнением некоторой программы. Поэтому и с процессом, и с потоком связывается о программный код в виде исполняемого модуля. Чтобы этот программный код был выполнен, его необходимо загрузить в ОП, выделить место на диске для хранения его данных, предоставить доступ к УВВ и т.д. В ходе выполнения программе также может понадобиться доступ к информационным ресурсам, например, файлам или семафорам. Естественно, программе для выполнения требуется и процессорное время. В ОС, поддерживающих процессы и потоки, процесс рассматривается как заявка на потребление всех видов ресурсов, кроме одного – процессора (процессорного времени). Этот главный ресурс распределяется ОС между другими единицами работы – потоками, получившими свое название потому, что они представляют собой последовательности (потоки выполнения) команд.

Процесс в общем случае может состоять из одного или нескольких потоков. В ОС, где у процесса может быть только один поток, понятие «поток» поглощается понятием «процесс», и остается одна единица работы и потребления ресурсов – процесс. Но в таких ОС возникают проблемы организации параллельных вычислений в рамках процесса. Ведь нередко приложение, выполняемое в рамках одного процесса, может обладать внутренним параллелизмом, который в принципе мог бы ускорить развитие процесса. Причем, чем больше параллельных ветвей содержится в приложении, тем больше его потенциал повышения производительности вычислений.

Потоки возникли в ОС именно как средство распараллеливания вычислений. Задачу распараллеливания вычислений можно решать следующими двумя традиционными способами.

1. Программист создает в приложении подпрограмму «диспетчер», периодически передающую управление каждой из параллельных ветвей вычислений. Но такая программа получится логически запутанной, с многочисленными передачами управления, что существенно затруднит ее отладку и модификацию. А при желании распараллелить все используемые приложения, такой способ вообще становится тупиковым.

2. Программист создает для одного приложения несколько процессов, по одному для каждой из параллельных работ. Но при использовании стандартных средств ОС нельзя учесть тот факт, эти процессы тесно взаимосвязаны (решают единую задачу, работают на одном общем поле данных), а поэтому должны иметь равные права на ресурсы ВС. А ОС будет рассматривать эти процессы наравне с другими и с помощью своих универсальных механизмов обеспечивать их изоляцию друг от друга. В данном случае эти полезные механизмы ОС будут выполнять, по сути, вредную для приложения работу, затрудняющую обмен данными между параллельными частями приложения. Кроме того, на создание каждого процесса ОС тратит определенные системные ресурсы, которые в данном случае неоправданно дублируются – каждому процессу выделяется собственное адресное пространство, ОП, закрепляются УВВ и т.п. Необходимо еще и согласованно обновлять общие данные.

Таким образом, в ОС наряду с процессами нужен *механизм распараллеливания вычислений*, учитывающий тесные связи отдельных ветвей приложения. Для этих целей современные ОС предлагают *механизм многопоточной обработки*.

Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время именно между потоками. ОС назначает процессу адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем порождение процессов. В отличие от процессов, принадлежащих разным конкурирующим приложениям, все потоки одного процесса принадлежат одному приложению. Поэтому ОС изолирует такие «родственные» потоки в гораздо меньшей степени. Все потоки одного процесса используют общие файлы, таймеры, УВВ, одну и ту же область ОП, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные, используют стеки друг друга. Для организации взаимодействия и обмена данными потокам не нужно обращаться к ОС, им достаточно использовать общую ОП. С другой стороны, «чужие» потоки разных процессов по-прежнему хорошо защищены средствами изоляции процессов.

Итак, мультипрограммирование более эффективно на уровне потоков, а не процессов. Каждый поток имеет собственный счетчик команд и стек. Задача, оформленная в виде нескольких потоков в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной ВС) выполнения отдельных ее частей.

Введение потоков выполнения упрощает программирование. Например, в задачах типа «писатель-читатель» один поток выполняет запись в буфер, другой считывает записи из него. Поскольку они разделяют общий буфер, нет смысла делать их отдельными процессами. Другой пример – использование потоков для управления сигналами прерывания от клавиатуры и других УВВ. Вместо обработки сигнала прерывания один поток специально назначается для постоянного ожидания поступления этих сигналов. Так можно сократить необходимость в прерываниях пользовательского уровня, повышая ясность программы.

Вследствие несамостоятельности своего создания поток имеет только 3 состояния: активно (выполнение), ожидание и готовность. В системе образуются очереди ожидающих и готовых к выполнению потоков.

24. Состав и краткая характеристика функциональных компонент ОС.

Функции ОС автономного компьютера обычно группируются либо в соответствии с видами локальных ресурсов, которыми управляет ОС, либо в соответствии со специфическими задачами, применимыми ко всем ресурсам. Обычно такие группы функций поддерживаются отдельными подсистемами ОС.

25. Функции подсистемы управления процессами.

Для каждого порождаемого процесса ОС генерирует системные информационные структуры, содержащие данные о потребностях процесса в ресурсах ВС, а также о фактически выделенных ему ресурсах. Таким образом, процесс действительно представляет собой заявку на потребление системных ресурсов.

Чтобы процесс был выполнен, ОС должна назначить ему область ОП, где будут размещены коды и данные процесса, а также предоставить ему необходимое количество процессорного времени. Кроме того, процессу может понадобиться доступ к другим ресурсам, например, к файлам или УВВ.

В информационные структуры процесса часто включаются вспомогательные данные, характеризующие историю пребывания процесса в системе (например, доли времени, потраченные процессом на вычисления и на обмен), его текущее состояние, значение приоритета. Эти данные могут учитываться ОС при принятии решения о предоставлении ресурсов данному процессу.

Важной задачей ОС является защита ресурсов, выделенных процессу, от остальных процессов. Одним из наиболее тщательно защищаемых ресурсов процесса являются области ОП, в которой хранятся коды и данные процесса. Совокупность всех областей ОП, которые ОС выделяет процессу, называется его **адресным пространством**. Каждый процесс работает в своем адресном пространстве, защищаемом ОС. Но ОС защищает и другие ресурсы (например, файлы, УВВ). А кроме защиты ресурсов еще и совместное их использование несколькими процессами, например, разрешать общий доступ к некоторой области ОП.

В реальных условиях выполнение процесса может быть многократно прервано и продолжено. Чтобы возобновить выполнение процесса, необходимо сохранить и, когда это будет нужно, восстановить состояние его операционной среды. Состояние операционной среды идентифицируется состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок

системных вызовов данного процесса и т.д. Эта важнейшая информация называется контекстом процесса. При смене процесса происходит переключение контекста.

ОС также выполняет функции синхронизации процессов, приостанавливая выполнение одного из них до наступления какого-либо события в системе, например, до завершения операции ввода-вывода, выполняемой ОС по его запросу.

Нет однозначного соответствия между процессами и программами. Один программный файл может породить несколько параллельных процессов, процесс может в ходе своего выполнения сменить программный файл и начать выполнять другую программу.

При реализации сложных программных комплексов их работу часто организуют в виде нескольких параллельных процессов, которые должны периодически взаимодействовать для обмена данными. Так как ОС защищает адресные пространства процессов по записи и чтению, то для оперативного взаимодействия процессов она должна предоставлять особые средства межпроцессного взаимодействия.

Таким образом, подсистема управления процессами планирует выполнение процессов, распределяя процессорное время между несколькими параллельными процессами, занимается порождением и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает синхронизацию процессов и взаимодействие между ними

26.Адресное пространство процесса, контекст процесса.

Адресное пространство (процесса) – совокупность всех областей ОП, которые ОС выделяет процессу.

Контекст процесса – состояние операционной среды, которое идентифицируется состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок системных вызовов данного процесса и т.д.

27.Функции подсистемы управления памятью.

Память является для процесса таким же важным и необходимым ресурсом, как и процессор. Процесс может стать активным только в том случае, если хотя бы часть его кодов и данных находится в ОП. Управление памятью включает:

1) Распределение имеющейся физической памяти между всеми существующими в системе процессами;
2) Загрузку кодов и данных процессов в отведенные им области памяти;

3) Настройку адресно-зависимых частей кодов процесса на физические адреса выделенной области памяти;
4) Защиту областей памяти каждого процесса.

Известно множество алгоритмов распределения ОП. Их отличием может быть, например:

- число выделяемых процессу областей памяти (одной непрерывной или нескольких несмежных);

- степень свободы расположения границы областей (статическая фиксация границы на всем ИСП или ее динамическое перемещение при дополнительном увеличении);

- единица и форма выделения памяти (страницами фиксированного размера или сегментами переменной длины).

Способом управления памятью является **механизм поддержки виртуальной памяти**, позволяющий программисту писать программы так, как будто в его распоряжении имеется однородная ОП достаточно большого размера.

Защита памяти – это избирательная способность ОС предохранять выполняемую задачу от записи или чтения памяти, назначенной какой-то другой задаче. Реальные программы часто содержат ошибки, вызывающие попытки обращения к «чужой» памяти. Средства защиты памяти в ОС должны пресекать несанкционированный доступ процессов к чужим областям памяти.

Таким образом, функциями ОС по управлению памятью являются:

- отслеживание свободной и занятой памяти;

- выделение памяти процессам и освобождение памяти при завершении процессов;

- защита памяти;

- вытеснение процессов из ОП на диск, если основной памяти недостаточно для размещения всех процессов, а также возвращение их обратно в ОП;

- настройка адресов программы на конкретную область физической памяти.

28. Функции подсистем управления файлами и УВВ, их взаимосвязь.

Способность ОС скрывать сложности реальной аппаратуры ярко проявляется в одной из основных подсистем ОС – **файловой системе** (ФС). ОС представляет отдельный набор данных, хранящийся на внешнем накопителе, в виде файла – простой неструктурированной последовательности байтов, имеющей символьное имя. Для удобства работы файлы группируются в многоуровневые каталоги.

Пользователь средствами ОС может выполнять над файлами и каталогами различные известные операции.

Чтобы представить множество наборов данных, разбросанных по цилиндрам и поверхностям дисков различных типов, в виде удобной иерархии файлов и каталогов, ОС должна решить множество задач. ФС ОС преобразует символьные имена файлов в физические адреса данных на диске, организует совместный доступ к файлам, защищает их от несанкционированного доступа.

При выполнении своих функций ФС тесно взаимодействует с подсистемой ввода-вывода (управления УВВ), которая по запросам ФС осуществляет передачу данных между дисками и ОП.

Подсистема ввода-вывода играет роль интерфейса ко всем УВВ компьютера. Спектр УВВ обширен, их номенклатура насчитывает сотни моделей. Эти модели могут существенно различаться набором и последовательностью команд обмена информацией с процессором и памятью компьютера, скоростью работы, кодировкой передаваемых данных, возможностью совместного использования и другими деталями.

Для управления конкретной моделью УВВ с учетом всех ее особенностей предназначен специальный драйвер. При этом драйвер может управлять единственной моделью или целой группой однотипных УВВ. Для пользователя очень важно, чтобы ОС включала как можно больше разнообразных драйверов, что гарантирует потенциальную возможность использования большего числа УВВ разных производителей. От наличия подходящих драйверов во многом зависит успех ОС на рынке.

Созданием драйверов УВВ занимаются как разработчики конкретной ОС, так и производители самих УВВ. А качественная ОС должна поддерживать хорошо определенный интерфейс между драйверами и остальной частью ОС, чтобы разработчики УВВ могли поставлять свои драйверы УВВ для данной ОС.

Прикладные программисты могут пользоваться интерфейсом драйверов при разработке своих программ. Но это не очень удобно, поскольку такой интерфейс обычно представляет собой низкоуровневые операции с огромным числом деталей.

Поэтому поддержание более удобного высокоуровневого унифицированного набора функций так называемого **интерфейса прикладного программирования** (Application Programming Interface, API) для разнородных УВВ является одной из наиболее важных задач ОС. Со времени появления ОС семейства UNIX такой унифицированный API в большинстве ОС строится на основе концепции файлового доступа. Ее суть состоит в том, что обмен с любым УВВ выглядит как обмен с файлом, имеющим имя и представляющим собой неструктурированную

последовательность байтов. В качестве файла может выступать реальный файл на диске или иное устройство, и это – самая удобная для пользователя или программиста абстракция.

29.Средства защиты данных и администрирования в ОС.

Безопасность данных ВС обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок ПО, а также средствами защиты от несанкционированного доступа, в частности, ошибочного или злонамеренного поведения пользователей системы.

Первое средство в защите данных от несанкционированного доступа:

– процедура логического входа (Login).

ОС должна убедиться, что в систему пытается войти пользователь, вход которого разрешен администратором системы. Функции защиты ОС тесно связаны с функциями администрирования, поскольку именно администратор определяет права всех пользователей при их обращении к ресурсам ВС. Администратор может и ограничивать возможности пользователей в выполнении тех или иных системных действий. Например, пользователю могут быть запрещены: выполнение процедуры завершения работы ОС, установка системного времени, завершение чужих процессов, создание учетных записей пользователей, изменение прав доступа к некоторым каталогам и файлам. Администратор может урезать даже возможности ГИП, ограничив доступ какого то пользователя к отдельным пунктам системы меню.

Важным средством защиты данных являются функции аудита ОС, состоящие в фиксации всех событий, от которых зависит безопасность системы. Это, например, попытки удачного и неудачного логического входа в систему, операции доступа к важнейшим каталогам и файлам, использование принтеров и другие. Список отслеживаемых событий определяет администратор ОС.

Поддержка отказоустойчивости реализуется ОС, как правило, на основе резервирования, например:

- поддерживается несколько копий данных на разных дисках или разных дисководах;

- резервируются принтеры и другие УВВ. При отказе одного из избыточных УВВ ОС должна быстро и прозрачным для пользователя образом реконфигурировать систему и продолжить работу с резервным устройством. Причем могут резервироваться как сами УВВ (например, жесткие диски), так и их контроллеры;

- могут резервироваться даже процессоры, и система продолжает работу при отказе одного из процессоров, хотя, возможно, и с меньшей производительностью;

- в пределе могут резервироваться даже целые серверы. Так образуется система с «зеркальными» серверами.

В состав ОС входят специальные программы, позволяющие администратору выполнять регулярные операции резервного копирования (Backup) для обеспечения быстрого восстановления (Restore) важных данных.

30. Роль и возможности API, концепции ГИП.

Прикладные программисты могут пользоваться интерфейсом драйверов при разработке своих программ. Но это не очень удобно, поскольку такой интерфейс обычно представляет собой низкоуровневые операции с огромным числом деталей.

Поэтому поддержание более удобного высокоуровневого унифицированного набора функций так называемого **интерфейса прикладного программирования** (Application Programming Interface, API).

Основу ГИП составляют следующие концепции.

1. Графическая модель интерфейса, обусловливающая использование графики и иконок, цвета и других средств для создания наглядного и конкретного интерфейса.
2. Концепция «рабочего стола» (Desktop), отражающая компьютеризованное представление информации и создание естественной пользовательской рабочей среды, похожей на поверхность обычного письменного стола.
3. Концепция окон (Windows). Окна обеспечивают параллельный вывод на экран взаимозависимой информации, когда все находится на виду у пользователя.
4. Объектная ориентация – фокусировка внимания пользователя в процессе диалога на видимые объекты интерфейса и доступные действия над этими объектами.
5. Фундаментальная концепция «point and select» (укажи и выбери). Дело здесь в том, что определить видимую альтернативу легче, чем вспомнить формат, синтаксис и параметры некоторой сложной команды; работать с устройствами управления позицией (мышью, клавиатурой) легче, чем набирать по памяти текст команд.
6. Направляемое (необязательное) главенство пользователя – мягкое управление его действиями (ненавязчивые советы и «подталкивания») и уменьшение количества информации, которую пользователю надо помнить.
7. Концепция согласованности (единообразия) интерфейса. У пользователя формируется устойчивая система ожидания одинаковых реакций ПС на его одинаковые действия, что формирует стереотип взаимодействия.

31. Принципы построения ГИП

Считается, что не существует правильного способа построения ГИП, но имеется несколько основополагающих принципов (руководящих начал), которые разработчик ГИП должен помнить. Рассмотрим эти принципы.

1) Пользователь почти всегда прав. Этот принцип подразумевает, что разработчик ГИП может и должен сделать многое во имя пользователя:
- хороший ГИП следует стратегии DWIM (Do What I Mean) – «я так и думал» или «если хочешь – сделай»;
- будет особенно удачно, если при ошибке пользователя ГИП оградит его от повторного совершения той же ошибки. К сожалению, на практике это реализуется далеко не всегда.

2) ПС должна быть удобной, а ГИП – естественным и дружественным.
3) Пользователь может разработать свою концептуальную модель интерфейса, то есть модель работы программы. А интерфейс должен поддерживать (покрывать) эту модель и обеспечить ожидаемый пользователем результат. При этом должны совпадать концептуальная модель (что я делаю, ожидаю и хочу получить) и ГИП (что я вижу, имею и чем могу управлять). Например, начинающие пользователи часто применяют тактику движения «на ощупь» в расчете на откат (Undo).
4) Пользователь может и должен управлять диалогом. А система должна позволить пользователю выполнять любые корректные действия в любой допустимой последовательности для решения его задачи. Прекрасные примеры таких возможностей демонстрирует работа в среде текстового процессора MS Office Word. Кроме того, пользователь должен иметь средства, которые позволят ему всегда владеть инициативой.
5) ГИП должен быть предсказуемым или даже интуитивным и учитывать:

1) здравый смысл (рисовальщик похож на карандаш, средство масштабирования – на линзу со знаками + и -, удачен инструмент заливки в виде малярного валика);

2) опыт взаимодействия с другими ПС (например, именно малярный валик везде). Конечно, от ГИП трудно требовать полной интуитивности. Например, Double Click и правая кнопка мыши не интуитивны. Если следовать этому принципу, то при работе с новой ПС уже можно будет предсказать ее поведение в какой-то новой для пользователя ситуации.

3) И если ваша догадка будет верной, то ГИП действительно был интуитивен. Более того, ГИП станет интуитивным для вас, как только вы навсегда запомните, как именно вам надо было работать. И чем меньше времени пройдет до этого момента, тем более удачен и интуитивен был ГИП.

6) Действия пользователя должны быть обратимыми, чтобы исключить потерю информации. Примеры: (1) действие «отмена» (Cancel) возвращает пользователя в предыдущее состояние; (2) действие «регенерация» (Refresh) восстанавливает измененные пользователем начальные значения на панели; (3) действие «откат» (Undo) отменяет эффект выполнения предыдущего ошибочного действия.
7) Контекст (информация, смысл которой позволяет уточнить значение входящих в нее действий) – обеспечивает поддержку пользовательской ориентации.

Примеры: панель заголовков, скроллинг – поддерживают контекст выводимой информации; перерисовка нового (активного) окна поверх старого поддерживает контекст текущего активного окна.

8) Наглядность ГИП с минимумом доверия к памяти пользователя. Пользователю доступны: (1) напоминания (помощь), (2) показ названий пунктов (возможных действий) в системе меню («Выбери, а не запоминай!»); (3) область функциональных клавиш (меню ключей) показывает текущее значение ключа, (4) дополнительные ключи – показываются рядом с действиями в пунктах меню.
9) Обратная (успокаивающая) связь для действий пользователя: (1) цвет, (2) выделение и специальные индикации отбора для выделения места и сделанного выбора; (3) резкий характерный звук, если действия пользователя приведут к неожиданным или неудачным результатам.
10) Необходимость подтверждения потенциально разрушительных действий с выводом разъясняющих ситуацию сообщений в окнах или опций.
11) Совместимость интерфейса с экраном и безоконными средами. Было время, когда окон на экране вообще не было. Так пусть же этой ситуации теперь соответствует распахнутое на весь экран окно.
12) Наличие альтернативных (обычных, ускоренных) средств выполнения многошаговых действий. Например, удобно использовать последовательности нажатия функциональных клавиш или пиктограммы. Еще лучше включить средства вывода на Desktop или программирования ускорителей для тех, кому это надо.
13) Делать эффективным интерфейс клавиатуры для наиболее часто встречающихся офисных приложений и ПС. Мышь не должна становиться необходимой, оставаясь удобной! Необходим дублирующий интерфейс клавиатуры, иначе при отказе или отсутствии мыши пользователь ничего вообще не сможет сделать.
14) Обеспечить поддержку совместного действия «мыши» и клавиатуры в любой точке диалога. И пусть пользователь сам выберет устройство, которое лучше подойдет ему на следующем шаге.
15) Стараться придерживаться совместимости и согласованности (единообразия), особенно фундаментальных понятий и отношений, во всех операционных средах.
16) Должны поддерживаться 3 вида согласованности ГИП на уровне средств и возможностей пользователя:
    - *семантическая*, когда согласуются значения (смысл) элементов ГИП (результат и смысл вызова некоторой команды). Она позволяет пользователю разрабатывать стратегии решения задач в одной ПС и затем применять их в других ПС;
    - *синтаксическая* – одинаковые значения должны обозначаться одними и теми же терминами. Она позволяет пользователю разрабатывать свои стратегии в обозначениях общих терминов, которые могут непосредственно применяться в целом ряде ПС;
    - *физическая* – при указании одинаковых действий должны использоваться одинаковые последовательности действий.

Разработчик ГИП отвечает за реализацию согласованности, необходимой пользователю. Решение этой задачи могут упростить вспомогательные интерфейсы, если они будут функционально поддерживать рассмотренные виды согласованности.

32. Технические приемы и дополнительные правила построения ГИП

Основу ГИП составляют следующие концепции.

1. Графическая модель интерфейса, обусловливающая использование графики и пиктограмм (иконок), цвета и других выразительных средств для создания наглядного и конкретного интерфейса.

2. Концепция «рабочего стола» (Desktop), отражающая компьютеризованное представление информации и создание естественной пользовательской рабочей среды, похожей на поверхность обычного письменного стола.

3. Концепция окон (Windows). Окна обеспечивают параллельный вывод на

экран взаимозависимой информации, когда все находится на виду у пользователя.

4. Объектная ориентация – фокусировка внимания пользователя в процессе диалога на видимые объекты интерфейса и доступные действия над этими объектами.

5. Фундаментальная концепция «point and select» (укажи и выбери). Дело здесь в том, что определить видимую альтернативу легче, чем вспомнить формат, синтаксис и параметры некоторой сложной команды; работать с устройствами управления позицией (мышью, клавиатурой) легче, чем набирать по памяти текст команд.

6. Направляемое (необязательное) главенство пользователя – мягкое управление его действиями (ненавязчивые советы и «подталкивания») и уменьшение количества информации, которую пользователю надо помнить.

7. Концепция согласованности (единообразия) интерфейса. У пользователя формируется устойчивая система ожидания одинаковых реакций ПС на его одинаковые действия, что формирует стереотип взаимодействия.

Имеются и некоторые объективные полезные дополнительные правила создания ГИП, которые может выявить для себя каждый разработчик. Рассмотрим лишь некоторые такие правила.

1. ГИП не должен бросаться в глаза. Примеры: элемент выбора из меню Help не должен появляться из-за нижнего края окна; кнопки должны быть серыми; шрифты должны быть стандартными системными.

2. ГИП должен иметь эстетическую привлекательность.

3. ГИП должен деликатно поддерживать пользовательскую технологию освоения ПС методом проб и ошибок: удачный выход из ситуаций потери ориентации; снижение вероятности ошибок – понятные варианты выбора; ясное и деликатное сообщение об ошибке.

4. ГИП должен предвосхищать события – следующие шаги пользователя (с предложением выполнить их, не дожидаясь, пока он сам догадается): запоминание (выделение) последнего выбранного элемента списка, состояния, значения и т.п., чтобы при возврате не вспоминать последнее выполненное действие; поля ввода с прямой навигацией, побуквенный фильтр вариантов выбора имен; запоминание имен файлов последнего использования в обратном порядке.

5. ГИП должен быть целостным и функционально полным.

6. Если при использовании вашего ГИП хоть один пользователь сделал ошибочное действие, то вам есть еще над чем работать. Например, название окна «Завершение работы с Windows…» лучше перевести как «Варианты завершения/продолжения...».

7. Если какое-то правило (или даже принцип) неприменимо или не имеет смысла, то хороший разработчик знает, когда можно его нарушить или изменить. Главное – понимать суть правила и цель его нарушения. Без этого пункта дальнейшее развитие и совершенствование ГИП было бы просто невозможным

33. Эволюция ГИП популярных ОС и их семейств.

Рождение и эволюция

Вернемся более чем на 30 лет в прошлое, во время, когда многих из нас еще не было на белом свете. 10 ноября 1983 года Microsoft впервые показывает публике Windows 1.0, а 20 ноября 1985 года запускает в производство. Особенностью новой ОС (или надстройки над ОС) было наличие окон, которые можно было расположить друг около друга и возможность управления мышкой. Да, графические интерфейсы существовали и до Windows и многие сейчас ведут споры, кто у кого украл идею создания графического интерфейса, но, конечно, сам факт воровства идеи никак не влияет на успех или провал ОС на рынке ПО. Компьютеры в 80-х – это узкоспециализированные устройства, они не продаются в магазине за углом, их редко встретишь дома. Большая часть пользователей – работники крупных фирм, которым приходится работать с большими объемами данных или текстами в WordStar, например, и Windows начинает появляться на мониторах. Возможно, тогда это еще не было понятно, но 1983 год можно по праву назвать началом новой эры в развитии компьютерной техники, ведь родился прадед самых популярных пользовательских ОС в мире, хоть и принят он

был прохладно. Принципиально новое взаимодействие человека и машины с помощью нового устройства «мышь» впоследствии станет нормой.

В 1987 году Микрософт выпускает Windows 2.0, в которой окна уже могут перекрывать друг друга, в отличии от версии 1.0. Для отображения интерфейса используется 16 цветов, и к окнам впервые применены термины «свернуть» и «развернуть».

Windows 3.0. Теперь пользователь мог любоваться 256-ю цветами, что однозначно делало работу более приятной, хотя основные элементы интерфейса остались без существенных изменений.

Революция

Именно таким громким словом стоит назвать появление кнопки «Пуск!» и «Рабочего стола» в Windows 95. Вместе с ними мы получили хорошо знакомые ярлыки, «Проводник», «Панель задач» и фантастические звуковые эффекты. Новая версия поддерживала 16-битный и 32-битный цвет. Можно быть совершенно уверенным в том, что любой современный пользователь ПК не запутался бы в Windows 95, если вдруг возникнет какая-то необычная необходимость поработать с этой ОС.

Интерфейсы Windows 98 и Windows 98 SE не представляли собой ничего принципиально нового и практически не отличались друг от друга и незначительно отличались от Windows 95.

Самый оригинальный интерфейс или «Привет, Боб!»

В 1995 году Microsoft предприняла попытку выпустить оболочку для ОС Windows с принципиально новым интерфейсом. После загрузки в Bob пользователь видел изображение дома с различными комнатами, внешний вид которых можно было настраивать под себя. В каждой комнате можно было размещать тематически подобранные ярлыки приложений, так что ещё в 1995 году мы могли услышать популярное нынче слово «скевоморфизм» (хоть его смысл и отличается немного).

Если сопоставить во времени изменения интерфейса операционных систем с актуальными способами ввода информации и их популярностью, то совершенно очевидно, что:

1. Windows 1.0 – пионер операционных систем с оконным графическим интерфейсом и возможностью управления с помощью мыши и клавиатуры.
2. Windows 3.0 – первая популярная операционная система Microsoft с графическим интерфейсом.

3. Windows 95-98-98SE-2000-ME-XP-Vista-7 закрепляют успех и позиции Microsoft на рынке ОС. В этом ряду стоит выделить, наверное, Windows Vista – в плане развития графического интерфейса она не выделяется из общего ряда, но прохладно принята общественностью.
4. Windows 8 – пионер операционных систем с оконным интерфейсом, «тайлами» и возможностью управления с помощью мыши, клавиатуры и сенсорного экрана (пальцами).

Windows 8 соединила в себе, кажется, невозможное, и, вероятно, не всем нравится эта комбинация интерфейсов. Но вернемся на три десятилетия в прошлое к первой версии Windows. Такой же холодный прием, такие же недовольные пользователи, которые позднее искренне полюбили новые версии этой чудесной ОС. Признаюсь, я не в восторге от Windows 8, но я верю и надеюсь, что у Microsoft получится исправить ошибки, реализовать все свои идеи и создать новую и удобную ОС. Ведь когда-то получилось.

34. Необходимость, роль и классы прерываний.

Назначение и типы прерываний

Концепция прерывания (interrupt, INT) берет свое начало с того момента, когда разработчики компьютеров поняли, что без прерываний в работе ВС не обойтись и это обстоятельство можно применить с пользой: сделать прерывание основным механизмом функционирования ОС.

Примеры необходимости прерываний: в ОП отсутствуют данные, необходимые активной задаче; произошло событие, когда что-то случилось, появилось или закончилось в системе; более приоритетной задаче требуется процессор.

Прерывания, по своей сути, являются движущей силой любой ОС. Например, периодические прерывания от таймера вызывают смену процессов в мультипрограммной ОС, а прерывания от УВВ управляют потоками данных, которыми ВС обменивается с внешним миром. Система прерываний переводит процессор на выполнение иного потока команд вместо текущего (исходного) потока с последующим возвратом к исполнению исходного кода.

Механизм обработки прерываний похож на механизм выполнения процедур (подпрограмм), хотя и имеет важное отличие. Переход к подпрограмме происходит в заранее определенных программистом точках программы часто в зависимости от значений некоторых данных. Прерывание же может происходить в любой точке потока команд программы.

Прерывание возникает либо из-за определенных внешних по отношению к процессу выполнения программы событий, либо при появлении конкретных непредвиденных аварийных ситуаций в процессе ее выполнения. Сходство прерываний с процедурами состоит в том, в обоих случаях выполняется некоторая подпрограмма, обрабатывающая возникшую специальную ситуацию, а затем продолжается выполнение основной ветви программы.

В зависимости от источника своего возникновения прерывания делятся на 3 класса: внешние, внутренние и программные.

1. Внешние (аппаратные) прерывания могут возникать в результате действий пользователя или поступления сигналов от аппаратуры, например, сигналов завершения операций ввода-вывода, вырабатываемых контроллерами УВВ или сигналов от датчиков управляемых компьютером технических объектов. Они являются асинхронными по отношению к потоку инструкций прерываемой программы, возникая между выполнением двух соседних инструкций. При этом система после обработки внешнего прерывания продолжает выполнение прерванного процесса, начиная со следующей инструкции.

2. Внутренние прерывания или исключения (exceptions – особые исключительные ситуации) происходят синхронно с выполнением 72 программы (внутри тактов команды) при появлении аварийной ситуации в ходе выполнения некоторой инструкции. Примеры исключений: деление на нуль, ошибки защиты памяти, обращение по несуществующему адресу, попытка исполнить привилегированную инструкцию в пользовательском режиме и т.п.

3. Программные прерывания, по сути, не являются «истинными» прерываниями. Программное прерывание возникает при выполнении особой команды процессора, имитирующей прерывание, то есть переход на новую последовательность инструкций.

Прерываниям приписывается свой приоритет, на основе которого они ранжируются по степени важности и срочности. Прерывания с равными приоритетами образуют свой уровень приоритета. Число уровней прерываний может быть любым.

Прерывания обычно обрабатываются модулями ОС, так как важные, а часто даже критические действия, выполняемые по прерыванию, относятся к управлению важнейшими разделяемыми ресурсами ВС (процессором, принтером, диском, таймером). Процедуры, вызываемые по прерываниям, называют обработчиками прерываний или процедурами обслуживания прерываний (Interrupt Service Routine). Аппаратные прерывания обрабатываются драйверами соответствующих УВВ, исключения – специальными модулями ядра, а программные прерывания – процедурами ОС, обслуживающими системные вызовы. Кроме этих модулей ОС

может иметь еще и отдельный диспетчер прерываний, координирующий работу различных обработчиков прерываний.

35. Механизм векторных аппаратных прерываний

Механизм прерываний поддерживается аппаратными средствами компьютера и программными средствами ОС. Аппаратная поддержка прерываний имеет свои особенности, зависящие от типа процессора и других аппаратных компонентов, передающих сигнал запроса прерывания от УВВ к процессору. Особенности аппаратной реализации прерываний влияют на средства их программной поддержки, работающие в составе ОС.

Существуют два основных способа, с помощью которых 73 выполняются аппаратные прерывания: векторный (vectored) и опрашиваемый (polled). В обоих способах процессору предоставляется информация об уровне приоритета прерывания на шине подключения УВВ. В случае векторных прерываний в процессор передается также информация о начальном адресе обработчика прерываний.

Каждому устройству, использующему векторные прерывания, назначается так называемый вектор прерывания – электрический сигнал, выставляемый на соответствующие шины процессора и несущий в себе информацию об определенном, закрепленном за данным устройством номере его обработчика прерываний. Этот вектор может быть фиксируемым, конфигурируемым (например, переключателями) или программируемым. ОС может предусматривать процедуру регистрации вектора (обработки) прерывания для определенного устройства, связывающую некоторый обработчик с каким-то определенным вектором. Вектор нужен процессору для нахождения обработчика запрашиваемого прерывания.

Вектор прерывания в информационном плане представляет собой 74 целое число (номер) в диапазоне 0-255, указывающее на одну из 256 программ обработки прерываний, адреса которых хранятся в таблице обработчиков прерываний. Когда к каждой линии IRQ подключается только одно устройство, процедура обработки прерываний работает по векторной схеме (без дополнительных опросов УВВ). Однако при совместном использовании одного уровня IRQ несколькими УВВ программа обработки прерываний должна работать по схеме опрашиваемых прерываний, дополнительно производя опрос всех УВВ, подключенных к каждому уровню IRQ.

Замечание о векторах прерываний. При описании механизма векторных прерываний разработчики ОС и авторы публикаций могут использовать различные термины, обозначающие близкие по смыслу понятия: таблица векторов (обработчиков) прерываний (или вектор прерываний) – массив данных в

памяти, содержащий для каждого номера прерывания адрес его обработчика; вектор прерывания – часть этой таблицы, относящаяся к конкретному прерыванию и включающая более одного поля для указания адреса соответствующего обработчика (поэтому в данном случае тоже используется термин «вектор»)

36. Механизм опрашиваемых аппаратных прерываний.

Механизм прерываний поддерживается аппаратными средствами компьютера и программными средствами ОС. Аппаратная поддержка прерываний имеет свои особенности, зависящие от типа процессора и других аппаратных компонентов, передающих сигнал запроса прерывания от УВВ к процессору. Особенности аппаратной реализации прерываний влияют на средства их программной поддержки, работающие в составе ОС.

Существуют два основных способа, с помощью которых выполняются аппаратные прерывания: векторный (vectored) и опрашиваемый (polled). В обоих способах процессору предоставляется информация об уровне приоритета прерывания на шине подключения УВВ. В случае векторных прерываний в процессор передается также информация о начальном адресе обработчика прерываний.

При использовании опрашиваемых прерываний процессор получает от запросившего прерывание устройства только уровень его приоритета прерывания. С каждым уровнем в общем случае может быть связано несколько устройств и соответственно несколько обработчиков прерываний. Поэтому при каждом запросе прерывания вызываются все обработчики данного уровня, пока один из них не подтвердит, что это прерывание пришло от обслуживаемого им устройства.

Но механизм прерываний некоторой аппаратной платформы может и сочетать векторный и опрашиваемый типы прерываний. Например, в ПК на платформе Intel Pentium шины PCI, ISA, EISA, используемые для подключения УВВ, поддерживают механизм опрашиваемых прерываний. Контроллеры УВВ выставляют на шину не вектор, а сигнал запроса прерывания определенного уровня IRQ. Однако в процессоре Pentium система прерываний является векторной. Поэтому контроллер прерываний предварительно отображает поступающий от шины сигнал IRQ на определенный номер вектора и подает в процессор Pentium уже вектор прерывания.

37. Приоритетность и маскирование прерываний.

Механизм прерываний чаще всего поддерживает приоритетность и маскирование прерываний. Приоритетность означает, что все источники прерываний делятся на классы и каждому классу назначается свой уровень

приоритета запроса на прерывание. Приоритеты могут обслуживаться как относительные или абсолютные. Обслуживание запросов прерываний по схеме с относительными приоритетами заключается в том, что при одновременном поступлении запросов из разных классов выбирается запрос, имеющий наивысший приоритет, и его обслуживание уже не прерывается, даже при поступлении более приоритетных запросов. При обслуживании на основе абсолютных приоритетов более приоритетный запрос прерывает обслуживание менее приоритетного и обрабатывается, после чего продолжается прерванное обслуживание.

В схеме абсолютных приоритетов также выполняется маскирование: на время обслуживания каждого запроса вводится запрет обслуживания запросов с равным или более низким приоритетом. Но схема 75 маскирования предполагает и возможность временного запрета обслуживания прерываний любого класса независимо от уровня приоритета.

38. Последовательность действий аппаратных и программных средств по обработке прерывания.

Обобщенно последовательность действий аппаратных и программных средств по обработке прерывания можно описать следующим образом.

1. При возникновении сигнала (для аппаратных прерываний) или условия (для внутренних прерываний) прерывания происходит первичное аппаратное распознавание типа прерывания. Если прерывания данного типа в настоящий момент запрещены (приоритетной схемой или механизмом маскирования), то процессор продолжает поддерживать естественный ход выполнения команд. В противном случае в зависимости от поступившей в процессор информации (уровень прерывания, вектор прерывания или тип условия внутреннего прерывания) происходит автоматический вызов процедуры обработки прерывания, адрес которой находится в специальной таблице операционной системы, размещаемой либо в регистрах процессора, либо в определенном месте оперативной памяти.

2. Автоматически сохраняется некоторая часть контекста прерванного потока, которая позволит ядру возобновить исполнение потока процесса после обработки прерывания. В это подмножество обычно включаются значения счетчика команд, слова состояния машины, хранящего признаки основных режимов работы процессора (пример такого слова — регистр EFLA6S в Intel Pentium), а также нескольких регистров общего назначения, которые требуются программе обработки прерывания. Может быть сохранен и полный контекст процесса, если ОС обслуживает данное прерывание со сменой процесса. Однако в

общем случае это не обязательно, часто обработка прерываний выполняется без вытеснения текущего процесса.

Решение о перепланировании процессов может быть принято в ходе обработки прерывания, например, если это прерывание от таймера и после наращивания значения системных часов выясняется, что процесс исчерпал выделенный ему квант времени. Однако это совсем не обязательно — прерывание может выполняться и без смены процесса, например прием очередной порции данных от контроллера внешнего устройства чаще всего происходит в рамках текущего процесса, хотя данные, скорее всего, предназначены другому процессу.

3. Одновременно с загрузкой адреса процедуры обработки прерываний в счетчик команд может автоматически выполняться загрузка нового значения слова состояния машины (или другой системной структуры, например селектора кодового сегмента в процессоре Pentium), которое определяет режимы работы процессора при обработке прерывания, в том числе работу в привилегированном режиме. В некоторых моделях процессоров переход в привилегированный режим за счет смены состояния машины при обработке прерывания является единственным способом смены режима. Прерывания практически во всех мультипрограммных ОС обрабатываются в привилегированном режиме модулями ядра, так как при этом обычно нужно выполнить ряд критических операций, от которых зависит жизнеспособность системы, — управлять внешними устройствами, перепланировать потоки и т. п.

4. Временно запрещаются прерывания данного типа, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры. Детали выполнения этой операции зависят от особенностей аппаратной платформы, например может использоваться механизм маскирования прерываний. Многие процессоры автоматически устанавливают признак запрета прерываний в начале цикла обработки прерывания, в противном случае это делает программа обработки прерываний.

5. После того как прерывание обработано ядром операционной системы, прерванный контекст восстанавливается и работа потока возобновляется с прерванного места. Часть контекста восстанавливается аппаратно по команде возврата из прерываний (например, адрес следующей команды и слово состояния машины), а часть — программным способом, с помощью явных команд извлечения данных из стека. При возврате из прерывания блокировка повторных прерываний данного типа снимается.

39. Программные прерывания

Программное прерывание реализует один из способов перехода на подпрограмму с помощью специальной команды (инструкции) процессора,

например, INT в процессорах Intel Pentium. При выполнении команды программного прерывания процессор отрабатывает ту же, рассмотренную выше последовательность действий, но только происходит это в предсказуемой точке программы – там, где программист поместил данную команду .

Практически все современные процессоры имеют в своей системе команд инструкции программных прерываний. Одной из причин их появления является то, что их использование часто приводит к более компактному коду программ по сравнению с использованием стандартных команд выполнения процедур. Это объясняется тем, что разработчики процессора обычно резервируют для обработки прерываний небольшое число возможных подпрограмм. Поэтому длина операнда в команде программного прерывания, который указывает на нужную подпрограмму, оказывается меньше, чем в команде CALL перехода на эту подпрограмму. 77

Например, в процессоре x86 предусмотрена возможность применения 256 программ обработки прерываний (256 = 28), поэтому в инструкции INT операнд имеет длину в 1 байт. Значение операнда команды INT просто является индексом в таблице из 256 адресов подпрограмм обработки прерываний, один из которых и используется для перехода по команде INT. А при использовании команды CALL потребовался бы уже код длиной в 2 или 4 байта. Другой причиной применения программных прерываний вместо обычных инструкций вызова подпрограмм является возможность смены пользовательского режима на привилегированный одновременно с вызовом процедуры.

В результате программные прерывания часто используются для выполнения ограниченного числа вызовов функций ядра ОС – системных вызовов.

40. Виртуализация. Примеры виртуализации.

Виртуальный ресурс – кажущийся, возможный, не имеющий физического воплощения или воспринимаемый иначе, чем он реализован.

Виртуализация – переход на более высокий уровень абстракции в управлении конкретными конфигурациями ВС. Построение виртуальных ресурсов, их распределение и использование свойственны любой ОС. Концепция виртуализации:й

- устраняет конфликты при управлении процессами и распределении ресурсов в условиях нехватки ресурсов;
- облегчает работу пользователя с системой;
- освобождает пользователя от рутинных процедур при обращении к ограниченным физическим ресурсам.

Для осуществления виртуализации нужна централизованная схема распределения ресурсов.

При виртуализации ресурсов получаются две формы «обмана» пользователей:

- обман 1: ресурс реально не существует или существует, но с ухудшенными характеристиками;
- обман 2: для нескольких параллельных процессов создается иллюзия одновременного использования того, что в реальной системе существовать не может.

Такой «обман» преследует вполне благородные цели – предоставить пользователям ресурсы с характеристиками, в наибольшей степени их устраивающими, и снять ограничение на количество распределяемых ресурсов, что позволит повысить скорость развития процессов и гибкость управления процессами в том или ином режиме мультипрограммной обработки .

Примеры виртуализации.

1. Построение на одном реальном таймере нескольких виртуальных – обман 2: для параллельных процессов используется много таймеров.

2. Построение на одном реальном канале нескольких виртуальных – обман 2: для параллельных процессов используется много каналов.

3. Виртуализация УВВ – обман 1: есть требуемое устройство. На базе реального УВВ с лучшими характеристиками моделируется работа УВВ с худшими характеристиками. Для организации виртуального флоппи-диска выбирается пространство реального жестокого диска, с помощью которого моделируется виртуальный флоппи-диск. Еще лучше виртуальный диск в ОП. Его достоинством является самая высокая скорость работы, а недостаток состоит в том, что он работает только до серьезного сбоя или перезагрузки ОС.

4. Нулевое устройство – обман 1: есть устройство для вывода. Это «заглушка» для псевдовывода в разных ОС (например, в UNIX).

5. Спулинг принтера – обман 2: для параллельных процессов есть много принтеров. Термин «спулинг» образован от английского simultaneous peripheral operation on-line (SPOOL). Принтер представляет последовательно используемый ресурс, который обычно замедляет развитие параллельных процессов, так как часто оказывается не вовремя занят. Реализация спулинга имеет следующие особенности :

- каждый процесс получает свое виртуальное устройство вывода;

- оно моделируется областью на жестком диске, то есть образуется буфер . При этом запись в буфер производится гораздо быстрее, чем вывод на принтер;
- фактическая выдача информации на единственный реальный принтер произойдет один раз, когда окончатся все процессы, связанные с данным буфером.

41. Виртуальная память и виртуальная машина

Виртуальная машина— программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы (target — целевая, или гостевая платформа) и исполняющая программы для target-платформы на host-платформе (host — хост-платформа, платформа-хозяин) или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы (см.: песочница); также спецификация некоторой вычислительной среды (например: «виртуальная машина языка программирования Си»).

Виртуальная машина исполняет некоторый машинно-независимый код (например, байт-код, шитый код, p-код) или машинный код реального процессора. Помимо процессора, ВМ может эмулировать работу как отдельных компонентов аппаратного обеспечения, так и целого реального компьютера (включая BIOS, оперативную память, жёсткий диск и другие периферийные устройства). В последнем случае в ВМ, как и на реальный компьютер, можно устанавливать операционные системы (например, Windows можно запускать в виртуальной машине под Linux или наоборот). На одном компьютере может функционировать несколько виртуальных машин (это может использоваться для имитации нескольких серверов на одном реальном сервере с целью оптимизации использования ресурсов сервера).

Виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (например, жёстким диском). Для выполняющейся программы данный метод полностью прозрачен и не требует дополнительных усилий со стороны программиста, однако реализация этого метода требует как аппаратной поддержки, так и поддержки со стороны операционной системы.

В системе с виртуальной памятью используемые программами адреса, называемые виртуальными адресами, транслируются в физические адреса в памяти компьютера. Трансляцию виртуальных адресов в физические выполняет аппаратное обеспечение, называемое блоком управления памятью. Для программы основная память выглядит как доступное и непрерывное адресное

пространство (англ.), либо как набор непрерывных сегментов, вне зависимости от наличия у компьютера соответствующего объёма оперативной памяти. Управление виртуальными адресными пространствами, соотнесение физической и виртуальной памяти, а также перемещение фрагментов памяти между основным и вторичным хранилищами выполняет операционная система (см. подкачка страниц).

Применение виртуальной памяти позволяет:

- освободить программиста от необходимости вручную управлять загрузкой частей программы в память и согласовывать использование памяти с другими программами
- предоставлять программам больше памяти, чем физически установлено в системе
- в многозадачных системах изолировать выполняющиеся программы друг от друга, путём назначения им непересекающихся адресных пространств (см. защита памяти)

В настоящее время виртуальная память аппаратно поддерживается в большинстве современных процессоров. В то же время в микроконтроллерах и в системах специального назначения, где требуется либо очень быстрая работа, либо есть ограничения на длительность отклика (системы реального времени) виртуальная память используется относительно редко. Также в таких системах реже встречается многозадачность и сложные иерархии памяти.

42. Дисциплина распределения ресурса, ее составляющие.

Сама идея мультипрограммирования непосредственно связана с наличием очередей процессов, так как процессы часто могут одновременно претендовать на одни и те же ресурсы. Если ресурс необходим и исчерпаем (дефицитен), то неизбежно придется его распределять, и будут возникать одна или несколько очередей запросов от процессов на данный ресурс. Процессор по очереди предоставляется процессам, очереди будут и при обращении к каналам, УВВ, модулям ОС. В обязанности ОС входит поддержание очередей запросов процессов на ресурсы.

Использование многими процессами того или иного последовательно используемого ресурса осуществляется с помощью некоторой дисциплины распределения ресурса (ДРР). Основой ДРР являются:

- дисциплина формирования очереди (ДФО) на ресурс – набор правил размещения запросов процессов в очереди;
- дисциплина обслуживания очереди (ДОО) – набор правил извлечения запроса процесса из очереди с последующим предоставлением ему ресурса для использования.

43. Одноочередные ДРР(Дисциплина распределения ресурса)

Вначале рассмотрим простейшие **одноочередные ДРР**, когда организуется единственная очередь запросов на ресурс от процессов.

1. Дисциплина обслуживания в порядке поступления или First In – First Out (FIFO).

2. Дисциплина обслуживания в порядке, обратном порядку поступления, Last In – First Out (LIFO), по сути представляющая стек.

В принципе возможна и очередь с двумя концами, имеющая название «дек» – от слов double ended queue (deque).

3. Круговой циклический алгоритм (КЦА) или карусельная (Round Robin) дисциплина. Здесь: **НЗ** – новый запрос, **ОЗ** – обслуженный запрос, **tk** – квант обслуживания. Данный алгоритм основан на дисциплине FIFO и используется при реализации режима РДВ.

В КЦА возможны две ситуации, когда могут быть:

- короткий НЗ, имеющий длительность облуживания t1≤ tk. Он обслуживается очень быстро (за 1 квант);
- длинный НЗ, его длительность облуживания t2 > tk (то есть 2 tk и более).

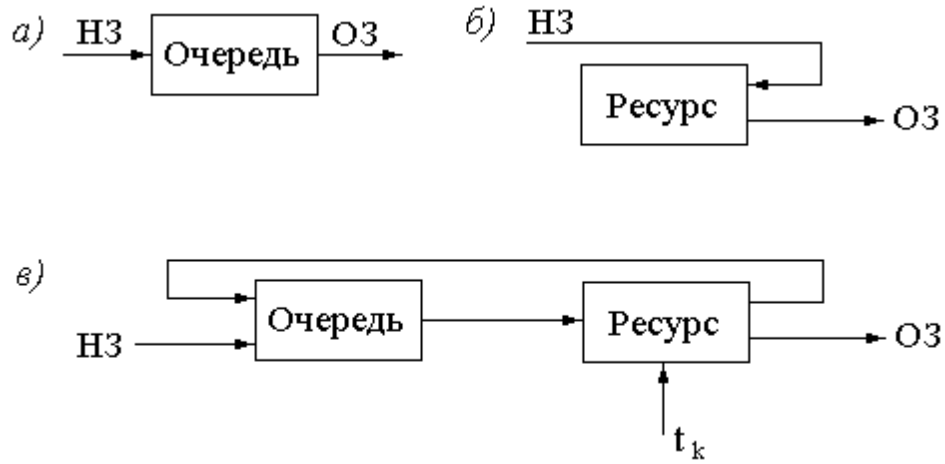


Рис.2.7. Дисциплины обслуживания FIFO, LIFO и КЦА

- Если измерять условную длину (**l**) запроса числом использованных **tk** и обозначить **ncp** – среднюю длину очереди, а $\mathbf{t_{срожид}}$ – среднее время ожидания, то

$$t_2 = l \times t_k + (l-1) \times t^{ср}_{ожид};$$

$$t^{ср}_{ожид} = t_k \times (n_{ср} - \quad 1),$$

откуда $$t_2 = t_k \times [n_{ср} \times (l-1) + 1].$$

В этой ситуации время обслуживания зависит от tk, длины запроса (l) и длины очереди (ncp).

4. Дисциплина обслуживания вида «следующим будет выполняться самый короткий запрос».

5. Дисциплина обслуживания вида «следующим будет выполняться запрос, которому осталось меньше всех других запросов выполняться на процессоре».

44. Многоочередные ДРР(Дисциплина распределения ресурса)

1. Неприоритетная многоочередная ДРР основана на КЦА:

- организуется N очередей;
- все новые запросы поступают в конец очереди 1;
- первый запрос очереди i (1≤i≤N) поступает на обслуживание, только если очереди 1, 2,…, (i-1) – пусты;
- на обслуживание выделяется квант времени tk;
- не до конца обслуженный запрос возвращается в конец очереди (i+1), что похоже на неявное снижение важности длинного запроса. Заметим, что не до конца обслуженный запрос из очереди N должен, в порядке исключения, возвращаться в конец этой последней очереди.

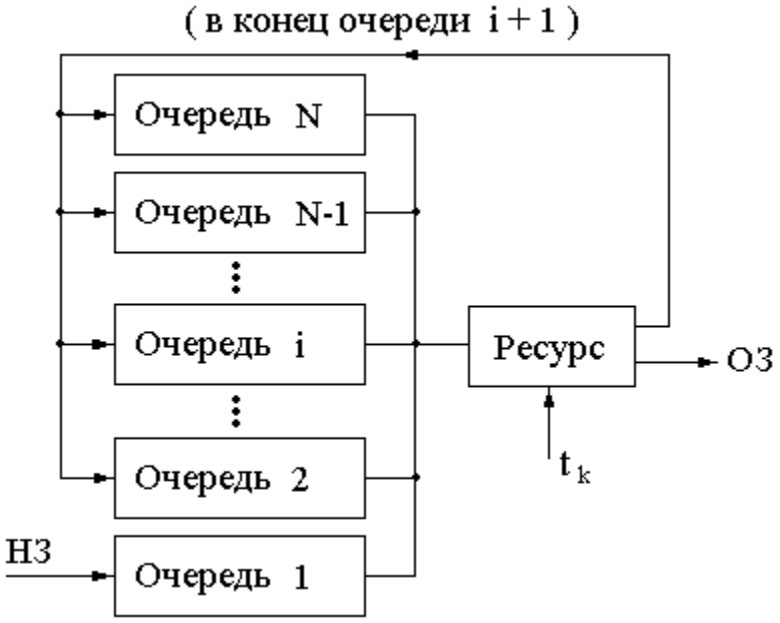


Рис.2.8. Схема неприоритетной многоочередной ДРР

Ее особенности: наиболее быстро обслуживаются короткие запросы; возникают непроизводительные затраты времени на перемещение не до конца обслуженных запросов между очередями; длинные запросы обслуживаются еще медленнее, чем в КЦА.

2. Приоритетная многоочередная ДРР (рис.2.9) также основана КЦА и является модификацией предыдущей дисциплины. Она отличается от предыдущей дисциплины тем, что каждый новый запрос имеет приоритет p=1÷N и попадает в очередь с номером j=N-p+1.

По отношению к новым запросам с приоритетами возможны две следующие стратегии поведения системы.

1) Обслуживание с абсолютным приоритетом. Если во время обслуживания запроса из очереди i поступает запрос в очередь j < i, то обслуживание i-го уровня прерывается, система начинает обслуживать более приоритетный запрос в течение **tk**. После окончания его обслуживания продолжается обслуживание прерванного запроса i-го уровня. Особенности: дискриминация низкоприоритетных запросов, время ожидания высокоприоритетных уменьшается, усложняется логика работы системы и ее реализация, появляется проблема прерывания обслуживания, и понадобятся дополнительные средства и ресурсы для ее реализации. Дисциплина подходит для систем управления объектами, в которых важна быстрая реакция на важное событие. Но при таком подходе возникают и сложные организационные проблемы:

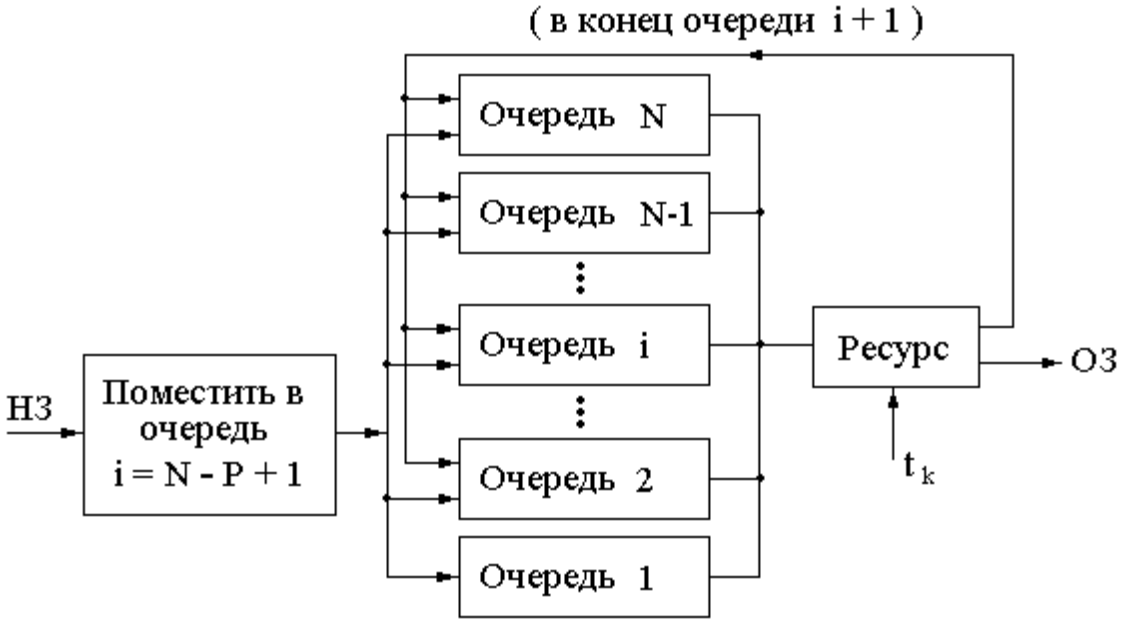


Рис.2.9. Схема приоритетной многоочередной ДРР

- найти удачное правило продолжения обслуживания прерванного запроса:
  - ❖ когда ему вновь выделять ресурс (если сразу – новые затраты);
  - ❖ учитывать, что ресурс уже использовался до прерывания, или нет;
- выбрать, в какую очередь помещать прерванный запрос (i+1, i, i-1);
- что делать, если во время прерывания обслуживания появится еще запрос уровня q < I (проблема маскирования)?

2) Обслуживание с относительным приоритетом. Новый запрос не вызывает прерывания обслуживания запроса, даже если обслуживается менее приоритетный запрос. Только после окончания обслуживания текущего (менее приоритетного) запроса начнется обслуживание нового (более приоритетного). Особенности: минимизируются затраты на переключения процессора, но допускается его монополизация. Дисциплина не подходит для систем РДВ и РВ.

45. Факторы, осложняющие в реальных условиях решение проблемы распределения ресурсов, в том числе на примере ОП.

Проблема распределения ресурсов в реальных условиях может оказаться гораздо сложнее, так как потребуется учет:

- возможной взаимосвязи процессов;
- стратегий распределения других ресурсов;
- тупиковых ситуаций, когда возникает циклический конфликт занятости ресурсов. Пример подобной ситуации приведен на рис.2.10, где Процессу 1 требуются Ресурсы 1 и 2, но выделяется только Ресурс 1, так как Ресурс 2 ранее уже был выделен Процессу 2, которому требовался еще Ресурс 1. Но Ресурс 1 также занят и не может быть выделен Процессу 2;
- прав процесса использовать разделяемый ресурс;
- специфики самого ресурса.

Кроме рассмотренных ДРР, существуют и другие, ориентированные на специфику самого распределяемого ресурса. Например, при распределении ОП используется:

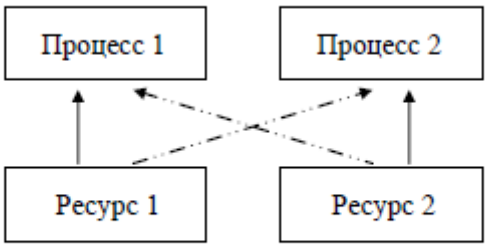


Рис.2.10. Пример тупиковой ситуации

- разная техника распределения ОП:
  - статическое распределение, когда до порождения процесса (заранее) создаются разделы ОП определенного размера, и каждая программа привязывается к своему разделу, только где она может быть запущена на решение; уменьшается гибкость, так как нет гарантии того, что программа поместится в данный раздел;

- динамическое распределение, каждый раздел создается в момент порождения процесса, и при этом часто потребуется ожидание освобождения ОП, поэтому возможны задержки, простои;
- комбинированное распределение, когда разделы создаются частично статически, частично динамически;

- разная структура ресурса (как простой ресурс – распределение непрерывными областями, или как составной – несмежными частями).

46. Явление фрагментации ОП, пример и способы борьбы.

Так, при распределении непрерывными областями может возникать явление фрагментации. Его суть состоит в том, что после многократного выделения и освобождения областей ОП остаются незаполненные области (НО) все меньшего размера – такие, что размера каждой из них на каком-то шаге уже будет недостаточно для загрузки программы, а суммарный размер свободной ОП будет по прежнему достаточно большим. Пример фрагментации приведен на рис.2.11,а–д, где показано распределение ОП для задач разного размера (в условных единицах в скобках) по состоянию в разные моменты времени t1, t2, …, t5.

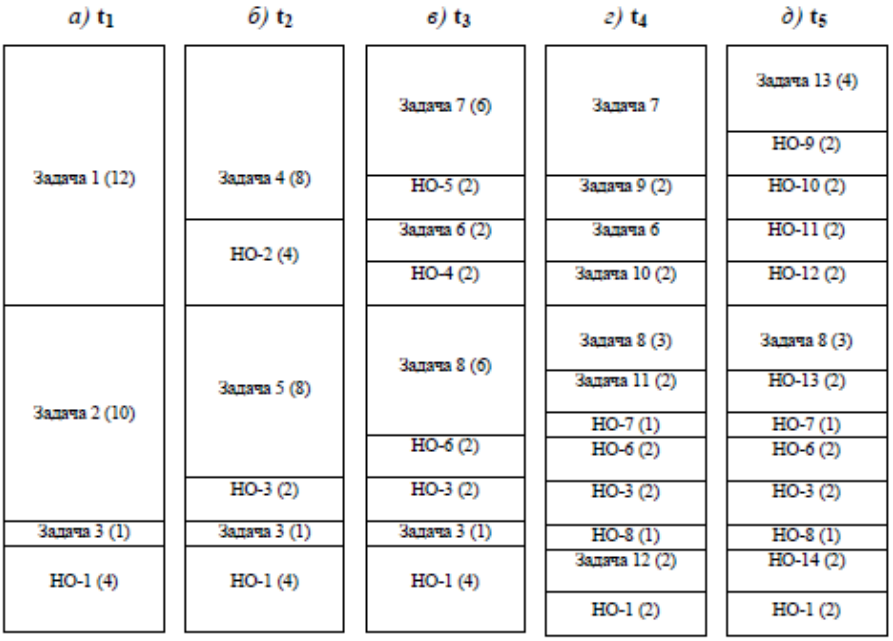


Рис.2.11. Распределение ОП для задач разного объема по состоянию моменты времени t_1, t_2, …, t_5.

Таблица 2.2

Момент времени:	$t_0=0$	t_1	t_2	t_3	t_4	t_5
Предел объема загружаемой задачи (усл.ед.) в момент t_i	27	4	4	4	2	2
Предел объема загружаемой задачи (усл.ед.) после t_i и завершения других задач	27	12	8	6	6	4

Из рис.2.11 видно, что в момент времени t5 в ОП не может быть загружена задача размером 3 единицы в то время, когда суммарный размер НО составляет 20 единиц из общего числа 27 единиц. Причем, ограничение размера загружаемых задач ужесточается с течением времени, что отражено в табл.2.2.

Поэтому при таком распределении ОП необходимы программные средства объединения всех малых НО ОП в единственную непрерывную область на основе различных хорошо известных алгоритмов устранения «дыр» в ОП.

47. Варианты организации динамических приоритетных циклических (карусельных) схем ДРР.

Кроме того, многоочередная ДРР может строиться на основе еще более сложных в организации динамических приоритетных циклических (карусельных) схем, сочетающих элементы режимов РВ и РДВ.

На рис.2.12 показана двухступенчатая схема подобного варианта, где в виде большого кольца изображена главная карусель (циклическая очередь задач к процессору). С ней связано еще несколько локальных каруселей (малых колец).

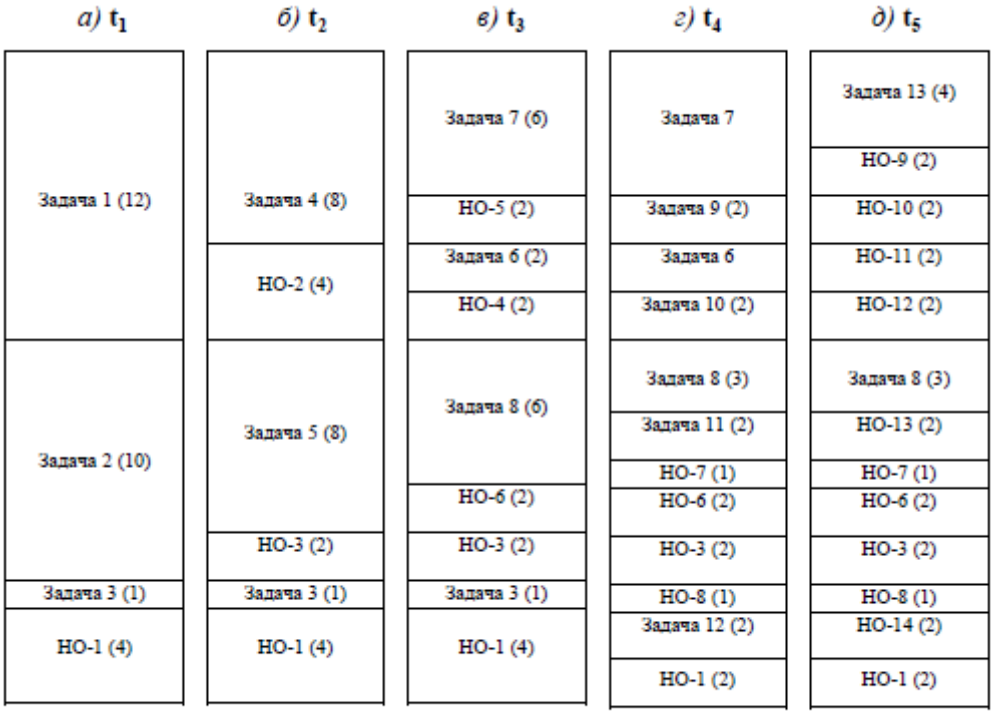


Рис.2.11. Распределение ОП для задач разного объема по состоянию моменты времени $t_1, t_2, \ldots, t_5$.

Таблица 2.2

Момент времени:	$t_0=0$	t_1	t_2	t_3	t_4	t_5
Предел объема загружаемой задачи (усл.ед.) в момент t_i	27	4	4	4	2	2
Предел объема загружаемой задачи (усл.ед.) после t_i и завершения других задач	27	12	8	6	6	4

Каждая задача, условно изображенная в виде квадрата, попадает в основную карусель из своей локальной, куда и возвращается при появлении в данной локальной карусели более приоритетной задачи.

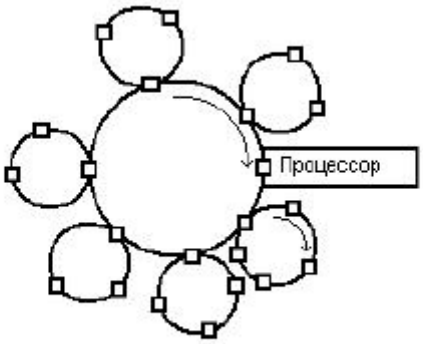


Рис.2.12. Двухступенчатая карусельная схема

Смена «представителя» в главной карусели организуется за счет динамического временного повышения или понижения приоритета задач в отдельной локальной карусели. В главной карусели предоставление процессора может осуществляться по аналогичному или какому-то иному, например, эвристическому правилу.

Карусельная схема может быть и многоступенчатой, как показано на рис.2.13, или сочетать два последних варианта.

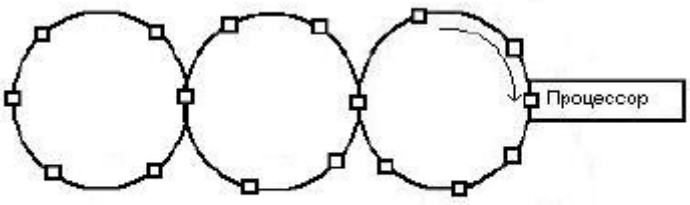


Рис.2.13. Многоступенчатая карусельная схема

48. Уровни взаимодействия пользователя с компьютером. Возможности командных языков ОС (с примерами).

Выделяют два вида взаимодействия пользователя с компьютером:

- элементарное (краткосрочное) взаимодействие. Оно осуществляется при выполнении конкретной программы и реализуется средствами стандартного ГИП ПС;
- комплексное (долгосрочное) взаимодействие. Оно проявляется при работе с заданием (пакетом), управлении операционной средой и ее настройке, планировании, администрировании и т.п. Комплексное взаимодействие обеспечивается средствами различных языков управления в ОС и является предметом нашего рассмотрения.

Для управления работой ВС и планирования выполнения процессов, (заданий, задач) ОС необходима определенная информация: требуемые объемы ОП для заданий и их отдельных шагов, приоритеты задач, типы УВВ, характеристики наборов данных и другая. Необходима конкретизация этой информации по определенным правилам, которые обычно представляются специальными языками: командными для диалоговых ОС или языками управления заданиями – для пакетных ОС.

Рассматриваемые языки описывают четыре группы операций.

1. Операции вызова для исполнения системных и пользовательских программ или командных (пакетных) файлов. Например, это директивы:

- RUN <имя программы>;
- EXECUTE <имя программы>;
- <имя программы>.

Программный модуль ОС, ответственный за отработку отдельных команд или их последовательности из пакетного файла, называется интерпретатором команд.

2. Действия по доступу к данным и их преобразованию. Характеристики данных задаются явно.

3. Операции управления заданиями (пакетными файлами). Может задаваться дополнительная информация: требуемые ресурсы, приоритеты, имена заданий и шагов, значения параметров и т.п.

4. Действия по настройке аппаратно-программной среды, администрирование.

Примеры командных языков:

- командный язык семейства ОС Windows NT/2000/XP/Vista/W7/W8 позволяет работать с пакетными файлами, поддерживать автозагрузку, а в последних версиях – еще и различные сценарии работы пользователя;
- командный язык ОС OS/2, кроме отмеченного выше, например, позволяет задать максимальное число поддерживаемых потоков в системе (команда

THREADS), значения различных параметров многозадачного режима, в частности, динамически изменяемую в определенных пределах величину кванта (команда TIMESLICE);

- командный язык ОС семейства UNIX выводит подобные языки на уровень развитых и достаточно сложных языков программирования. Его широкие возможности должен знать и обычно активно применяет каждый пользователь, без знания его конструкций нельзя даже включить в работу (смонтировать) дисковод, диджитайзер, или иное УВВ.

49. Задачи обеспечения безопасности в ОС. Внутренние и внешние угрозы безопасности.

Безопасность данных ВС обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок ПО, а также средствами защиты от несанкционированного доступа, в частности, ошибочного или злонамеренного поведения пользователей системы.

50. Объекты и субъекты защиты, методы идентификации и аутентификации пользователя.

Первое средство в защите данных от несанкционированного доступа – процедура логического входа (Login). ОС должна убедиться, что в систему пытается войти пользователь, вход которого разрешен администратором системы. Функции защиты ОС тесно связаны с функциями администрирования, поскольку именно администратор определяет права всех пользователей при их обращении к ресурсам ВС. Администратор может и ограничивать возможности пользователей в выполнении тех или иных системных действий. Например, пользователю могут быть запрещены: выполнение процедуры завершения работы ОС, установка системного времени, завершение чужих процессов, создание учетных записей пользователей, изменение прав доступа к некоторым каталогам и файлам. Администратор может урезать даже возможности ГИП, ограничив доступ какого-то пользователя к отдельным пунктам системы меню. Важным средством защиты данных являются функции аудита ОС, состоящие в фиксации всех событий, от которых зависит безопасность системы. Это, например, попытки удачного и неудачного логического входа в систему, операции доступа к важнейшим каталогам и файлам, использование принтеров и другие. Список отслеживаемых событий определяет администратор ОС.

51. Основные механизмы защиты.

Поддержка отказоустойчивости реализуется ОС, как правило, на основе резервирования, например:

- поддерживается несколько копий данных на разных дисках или разных дисководах;

- резервируются принтеры и другие УВВ. При отказе одного из избыточных УВВ ОС должна быстро и прозрачным для пользователя образом реконфигурировать систему и продолжить работу с резервным устройством. Причем могут резервироваться как сами УВВ (например, жесткие диски), так и их контроллеры;
- могут резервироваться даже процессоры, и система продолжает работу при отказе одного из процессоров, хотя, возможно, и с меньшей производительностью;
- в пределе могут резервироваться даже целые серверы. Так образуется система с «зеркальными» серверами.

52. Модели управления доступом и многоуровневой защиты.

Мандатное управление доступом (англ. *Mandatory access control, MAC*) — разграничение доступа субъектов к объектам, основанное на назначении метки конфиденциальности для информации, содержащейся в объектах, и выдаче официальных разрешений (допуска) субъектам на обращение к информации такого уровня конфиденциальности. Также иногда переводится как **Принудительный контроль доступа**. Это способ, сочетающий защиту и ограничение прав, применяемый по отношению к компьютерным процессам, данным и системным устройствам и предназначенный для предотвращения их нежелательного использования.

Согласно требованиям ФСТЭК мандатное управление доступом или «метки доступа» являются ключевым отличием систем защиты Государственной Тайны РФ старших классов 1В и 1Б от младших классов защитных систем на классическом разделении прав по матрице доступа.

Пример: субъект «Пользователь № 2», имеющий допуск уровня «не секретно», не может получить доступ к объекту, имеющему метку «для служебного пользования». В то же время, субъект «Пользователь № 1» с допуском уровня «секретно» право доступа к объекту с меткой «для служебного пользования» имеет.

Особенности

Мандатная модель управления доступом, помимо дискреционной и ролевой, является основой реализации разграничительной политики доступа к ресурсам при защите информации ограниченного доступа. При этом данная модель доступа практически не используется «в чистом виде», обычно на практике она дополняется элементами других моделей доступа.
Для файловых систем, оно может расширять или заменять дискреционный контроль доступа и концепцию пользователей и групп.
Самое важное достоинство заключается в том, что пользователь не может полностью управлять доступом к ресурсам, которые он создаёт.

Политика безопасности системы, установленная администратором, полностью определяет доступ, и обычно пользователю не разрешается устанавливать более свободный доступ к его ресурсам чем тот, который установлен администратором пользователю. Системы с дискреционным

контролем доступа разрешают пользователям полностью определять доступность их ресурсов, что означает, что они могут случайно или преднамеренно передать доступ неавторизованным пользователям. Такая система запрещает пользователю или процессу, обладающему определённым уровнем доверия, получать доступ к информации, процессам или устройствам более защищённого уровня. Тем самым обеспечивается изоляция пользователей и процессов, как известных, так и неизвестных системе (неизвестная программа должна быть максимально лишена доверия, и её доступ к устройствам и файлам должен ограничиваться сильнее).

Очевидно, что система, которая обеспечивает разделение данных и операций в компьютере, должна быть построена таким образом, чтобы её нельзя было «обойти». Она также должна давать возможность оценивать полезность и эффективность используемых правил и быть защищённой от постороннего вмешательства.

Поддержка в современных операционных системах

Изначально такой принцип был воплощён в операционных системах Flask, и других ориентированных на безопасность операционных системах.
Исследовательский проект АНБ SELinux добавил архитектуру мандатного контроля доступа к ядру Linux, и позднее был внесён в главную ветвь разработки в Августе 2003 года.
Мандатная система разграничения доступа реализована в ОС FreeBSD Unix.
В SUSE Linux и Ubuntu есть архитектура мандатного контроля доступа под названием AppArmor.

В сертифицированной в системах сертификации Минобороны России и ФСТЭК России операционной системе специального назначения Astra Linux Special Edition, механизм мандатного разграничения доступа реализован, как и механизм дискреционного разграничения доступа в ядре ОС и СУБД. Решение о запрете или разрешении доступа субъекта к объекту принимается на основе типа операции (чтение/запись/исполнение), мандатного контекста безопасности, связанного с каждым субъектом, и мандатной метки, связанной с объектом.
В сетевые пакеты протокола IPv4 в соответствии со стандартом RFC1108 внедряются мандатные метки, соответствующие метке объекта — сетевое соединение. В защищенных комплексах гипертекстовой обработки данных, электронной почты и в других сервисах, мандатное разграничение реализовано на основе программного интерфейса библиотек подсистемы безопасности PARSEC.

Управление доступом на основе ролей (англ. *Role Based Access Control, RBAC*) — развитие политики избирательного управления доступом, при этом права доступа субъектов системы на объекты группируются с учётом специфики их применения, образуя роли.
Формирование ролей призвано определить четкие и понятные для пользователей компьютерной системы правила разграничения доступа. Ролевое разграничение доступа позволяет реализовать гибкие, изменяющиеся динамически в процессе функционирования компьютерной системы правила разграничения доступа.

Такое разграничение доступа является составляющей многих современных компьютерных систем. Как правило, данный подход применяется в системах защиты СУБД, а отдельные элементы реализуются в сетевых операционных системах. Ролевой подход часто используется в системах, для пользователей которых четко определён круг их должностных полномочий и обязанностей.

Несмотря на то, что *Роль* является совокупностью прав доступа на объекты компьютерной системы, ролевое управление доступом отнюдь не является частным случаем избирательного управления доступом, так как его правила определяют порядок предоставления доступа субъектам компьютерной системы в зависимости от имеющихся (или отсутствующих) у него ролей в каждый момент времени, что является характерным для систем мандатного управления доступом. С другой стороны, правила ролевого разграничения доступа являются более гибкими, чем при мандатном подходе к разграничению.

Так как привилегии не назначаются пользователям непосредственно, и приобретаются ими только через свою роль (или роли), управление индивидуальными правами пользователя по сути сводится к назначению ему ролей. Это упрощает такие операции, как добавление пользователя или смена подразделения пользователем.

Технология управления доступом на основе ролей достаточно гибка и сильна, чтобы смоделировать как избирательное управление доступом (DAC), так и мандатное управление доступом (MAC)

До разработки RBAC, единственными известными моделями управления доступом были MAC и DAC: если модель была не MAC, то она была DAC, и наоборот. Исследования в 90-х показали, что RBAC не попадает ни в ту, ни в другую категорию.

Роли создаются внутри организации для различных рабочих функций. Определённым ролям присваиваются полномочия (permissions) для выполнения тех или иных операций. Штатным сотрудникам (или другим пользователям системы) назначаются фиксированные роли, через которые они получают соответствующие привилегии для выполнения фиксированных системных функций. В отличие от управления доступом на основе контекста, реализация RBAC в чистом виде не принимает во внимание текущую ситуацию (такую как, например, откуда было установлено соединение).

RBAC отличается от списков контроля доступа, используемых в традиционных избирательных системах управления доступом, тем, что может давать привилегии на сложные операции с составными данными, а не только на атомарные операции с низкоуровневыми объектами данных. Например, список контроля доступа может предоставить или лишить права записи в такой-то системный файл, но он не может ограничить то, каким образом этот файл может быть изменен. Система, основанная на RBAC, позволяет создать такую операцию как открытие «кредита» в финансовом приложении или заполнение записи «тест на уровень сахара в крови» в медицинском приложении. Присвоение привилегии на выполнение какой-либо операции многозначно, так как операции являются дробящимися в пределах приложения.

Концепции иерархии ролей и ограничений позволяют создать или смоделировать контроль доступа на основе решетки средствами RBAC. Таким образом, RBAC может быть основанием и расширением LBAC.

В организациях с разнородной IT-инфраструктурой, содержащих десятки и сотни систем и приложений, помогает использование иерархии ролей и наследования привилегий. Без этого использование RBAC становится крайне запутанным. В статье «Дополнительные роли: практический подход к обслуживанию пользователей предприятия» обсуждаются стратегии, альтернативные большому масштабу присвоения привилегий пользователям.

Современные системы расширяют старую модель NIST ограничениями RBAC для развертывания на больших предприятиях.

Для больших систем с сотнями ролей, тысячами пользователей и миллионами разрешений, управление ролями, пользователями, разрешениями и их взаимосвязями является сложной задачей, которую нереально выполнить малой группой администраторов безопасности. Привлекательной возможностью является использование самой RBAC для содействия децентрализованному управлению RBAC.

RBAC широко используется для управления пользовательскими привилегиями в пределах единой системы или приложения. Список таких систем включает в себя Microsoft Active Directory, SELinux, FreeBSD, Solaris, СУБД Oracle, PostgreSQL 8.1, SAP R/3, Lotus Notes и множество других.

53. Скрытые каналы передачи информации, классы безопасности информационных систем.

В критериях впервые введены четыре уровня доверия — D, C, B и A, которые подразделяются на классы. Классов безопасности всего шесть — C1, C2, B1, B2, B3, A1 (перечислены в порядке ужесточения требований).

Уровень D

Данный уровень предназначен для систем, признанных неудовлетворительными.

Уровень C

Иначе — произвольное управление доступом.

Класс C1

Политика безопасности и уровень гарантированности для данного класса должны удовлетворять следующим важнейшим требованиям:

1. доверенная вычислительная база должна управлять доступом именованных пользователей к именованным объектам;
2. пользователи должны идентифицировать себя, причем аутентификационная информация должна быть защищена от несанкционированного доступа;
3. доверенная вычислительная база должна поддерживать область для собственного выполнения, защищенную от внешних воздействий;
4. должны быть в наличии аппаратные или программные средства, позволяющие периодически проверять корректность функционирования

аппаратных и микропрограммных компонентов доверенной вычислительной базы;

5. защитные механизмы должны быть протестированы (нет способов обойти или разрушить средства защиты доверенной вычислительной базы);
6. должны быть описаны подход к безопасности и его применение при реализации доверенной вычислительной базы.

Класс C2

В дополнение к C1:

1. права доступа должны гранулироваться с точностью до пользователя. Все объекты должны подвергаться контролю доступа.
2. при выделении хранимого объекта из пула ресурсов доверенной вычислительной базы необходимо ликвидировать все следы его использования.
3. каждый пользователь системы должен уникальным образом идентифицироваться. Каждое регистрируемое действие должно ассоциироваться с конкретным пользователем.
4. доверенная вычислительная база должна создавать, поддерживать и защищать журнал регистрационной информации, относящейся к доступу к объектам, контролируемым базой.
5. тестирование должно подтвердить отсутствие очевидных недостатков в механизмах изоляции ресурсов и защиты регистрационной информации.

Уровень B

Также именуется — принудительное управление доступом.

Класс B1

В дополнение к C2:

1. доверенная вычислительная база должна управлять метками безопасности, ассоциируемыми с каждым субъектом и хранимым объектом.
2. доверенная вычислительная база должна обеспечить реализацию принудительного управления доступом всех субъектов ко всем хранимым объектам.
3. доверенная вычислительная база должна обеспечивать взаимную изоляцию процессов путем разделения их адресных пространств.
4. группа специалистов, полностью понимающих реализацию доверенной вычислительной базы, должна подвергнуть описание архитектуры, исходные и объектные коды тщательному анализу и тестированию.

5. должна существовать неформальная или формальная модель политики безопасности, поддерживаемой доверенной вычислительной базой.

Класс B2

В дополнение к B1:

1. снабжаться метками должны все ресурсы системы (например, ПЗУ), прямо или косвенно доступные субъектам.
2. к доверенной вычислительной базе должен поддерживаться доверенный коммуникационный путь для пользователя, выполняющего операции начальной идентификации и аутентификации.
3. должна быть предусмотрена возможность регистрации событий, связанных с организацией тайных каналов обмена с памятью.
4. доверенная вычислительная база должна быть внутренне структурирована на хорошо определенные, относительно независимые модули.
5. системный архитектор должен тщательно проанализировать возможности организации тайных каналов обмена с памятью и оценить максимальную пропускную способность каждого выявленного канала.
6. должна быть продемонстрирована относительная устойчивость доверенной вычислительной базы к попыткам проникновения.
7. модель политики безопасности должна быть формальной. Для доверенной вычислительной базы должны существовать описательные спецификации верхнего уровня, точно и полно определяющие её интерфейс.
8. в процессе разработки и сопровождения доверенной вычислительной базы должна использоваться система конфигурационного управления, обеспечивающая контроль изменений в описательных спецификациях верхнего уровня, иных архитектурных данных, реализационной документации, исходных текстах, работающей версии объектного кода, тестовых данных и документации.
9. тесты должны подтверждать действенность мер по уменьшению пропускной способности тайных каналов передачи информации.

Класс B3

В дополнение к B2:

1. для произвольного управления доступом должны обязательно использоваться списки управления доступом с указанием разрешенных режимов.
2. должна быть предусмотрена возможность регистрации появления или накопления событий, несущих угрозу политике безопасности системы.

Администратор безопасности должен немедленно извещаться о попытках нарушения политики безопасности, а система, в случае продолжения попыток, должна пресекать их наименее болезненным способом.

3. доверенная вычислительная база должна быть спроектирована и структурирована таким образом, чтобы использовать полный и концептуально простой защитный механизм с точно определенной семантикой.
4. процедура анализа должна быть выполнена для временных тайных каналов.
5. должна быть специфицирована роль администратора безопасности. Получить права администратора безопасности можно только после выполнения явных, протоколируемых действий.
6. должны существовать процедуры и/или механизмы, позволяющие произвести восстановление после сбоя или иного нарушения работы без ослабления защиты.
7. должна быть продемонстрирована устойчивость доверенной вычислительной базы к попыткам проникновения.

Уровень A

Носит название — верифицируемая безопасность.

Класс A1

В дополнение к B3:

1. тестирование должно продемонстрировать, что реализация доверенной вычислительной базы соответствует формальным спецификациям верхнего уровня.
2. помимо описательных, должны быть представлены формальные спецификации верхнего уровня. Необходимо использовать современные методы формальной спецификации и верификации систем.
3. механизм конфигурационного управления должен распространяться на весь жизненный цикл и все компоненты системы, имеющие отношение к обеспечению безопасности.
4. должно быть описано соответствие между формальными спецификациями верхнего уровня и исходными текстами.

54. Особенности защиты данных в ОС MS Windows и UNIX.

Одним из самых приоритетных направлений развития Майкрософт продолжает оставаться стратегия создания защищенных информационных систем, которая представляет собой взгляд на комплексное обеспечение информационной безопасности. Ключевыми моментами, определяющими успешность создания

защищенных систем, являются сотрудничество с государством и выполнение национальных требований, предъявляемых для обеспечения безопасности критически важных элементов информационной структуры.

Windows XP

Популярная пользовательская операционная система (ОС) со встроенными, сертифицированными по требованиям безопасности, средствами защиты информации, позволяющими использование данной ОС для защиты конфиденциальной информации и персональных данных, как на АРМ в составе сети, так и на автономных АРМ.

Unix

Первоначально разработанный как чисто исследовательское системное окружение, Unix в первое время располагал только рудиментарными возможностями защиты информации. Упор делался вовсе не на безопасность пользовательских приложений, а на гибкость, открытость и максимальное удобство среды разработки.

Благодаря этим достоинствам ОС Unix и стала столь популярной и широко распространенной операционной системой.

С другой стороны, вопросы защиты информации сегодня попадают в центр внимания правительств, деловых кругов и производителей. Естественно, особое значение вопросам безопасности данных придают правительства и военные. Так, в США за несколько последних лет принята масса федеральных законов, правительственных постановлений, директив различных агентств, ведомственных протоколов и технических спецификаций, посвященных этой проблеме. Коммерческие потребители также осознали необходимость защищать ценную информацию.

Рост популярности Unix и все большая осведомленность о проблемах безопасности привели к осознанию необходимости достичь приемлемого уровня безопасности ОС, сохранив при этом мобильность, гибкость и открытость программных продуктов. Уровень безопасности, который должен удовлетворять нуждам правительственных учреждений и коммерческих потребителей, закреплен в различных стандартах.

55. Методы шифрования

- ***Симметричное шифрование*** использует один и тот же ключ и для зашифрования, и для расшифрования.
- ***Асимметричное шифрование*** использует два разных ключа: один для зашифрования (который также называется открытым), другой для расшифрования (называется закрытым).

Эти методы решают определенные задачи и обладают как достоинствами, так и недостатками. Конкретный выбор применяемого метода зависит от целей, с которыми информация подвергается шифрованию.

Гибри́дная (или комбини́рованная) криптосисте́ма — это система шифрования, совмещающая преимущества криптосистемы с открытым ключом с производительностью симметричных криптосистем. Симметричный ключ используется для шифрования данных, а асимметричный для шифрования самого симметричного ключа, иначе это называется числовой упаковкой.

Криптографические системы используют преимущества двух основных криптосистем: симметричной и асимметричной криптографии. На этом принципе построены такие протоколы, как PGP и TLS.

Основной недостаток асимметричной криптографии состоит в низкой скорости из-за сложных вычислений, требуемых ее алгоритмами, в то время как симметричная криптография традиционно показывает высокую скорость работы. Однако симметричные криптосистемы имеет один существенный недостаток — её использование предполагает наличие защищенного канала для передачи ключей. Для преодоления этого недостатка прибегают к асимметричным криптосистемам, которые используют пару ключей: открытый и закрытый.

56. Цифровые подписи и сертификаты.

Цифровая подпись — это электронная зашифрованная печать, удостоверяющая подлинность цифровых данных, таких как сообщения электронной почты, макросы или электронные документы. Подпись подтверждает, что сведения предоставлены подписавшим их создателем и не были изменены.

Сертификат подписи.

Чтобы создать цифровую подпись, нужен сертификат подписи, удостоверяющий личность. Вместе с макросом или документом, заверенным цифровой подписью, также отправляется сертификат и открытый ключ. Сертификаты выпускаются центром сертификации и, аналогично водительскому удостоверению, могут быть отозваны. Как правило, сертификат действителен в течение года, по истечении которого подписывающий должен продлить его или получить новый сертификат для удостоверения своей личности.

57. Виды инсайдерских атак.

Неправомочное разглашение

В результате неправомочного разглашения или утечки важная для компании информация покидает корпоративный периметр и, в конце концов, попадает к лицам, у которых нет прав на доступ к этим данным и их использование. Это могут быть коммерческие и промышленные секреты фирмы, интеллектуальная собственность, персональные данные служащих, клиентов и партнеров, а также другие сведения, которые согласно проведенной классификации не являются публичными.

Украсть по неосторожности

Довольно часто инсайдеры подвергают корпоративные секреты риску непреднамеренно. Например, они могут случайно выложить секретные документы на веб-сайт, записать данные на ноутбук или карманный компьютер, который впоследствии будет украден или потерян, а также отослать конфиденциальные сведения по неверному почтовому адресу. Здесь могут использоваться те же самые средства защиты (фильтрация трафика и контроль операций на уровне рабочих станций) плюс шифрование данных на мобильных устройствах. Важно заметить, что, столкнувшись с невозможностью осуществить задуманную операцию (например, выложить документ в интернет или отослать по почте), такой инсайдер не станет упорствовать и, вероятно, обратится за помощью к коллегам, которые разъяснят неправомочность выполняемых действий. Именно этим, согласно экосистеме внутренних нарушителей, халатные и манипулируемые инсайдеры отличаются от обиженных и нелояльных.

Нарушение авторских прав на информацию

В рамках данной угрозы инсайдеры могут реализовать целый букет неправомерных действий. Например, скопировать части документов одного автора в документы другого автора (а также в почтовые сообщения, формы и т.д.), произвести индивидуальное шифрование документов, при котором компания лишается возможности работать с документом в случае утраты пароля или после увольнения сотрудника. Кроме того, инсайдеры могут использовать опубликованные в интернет материалы в своих документах без обработки, использовать мультимедиа-файлы (графику, аудио- и видеозаписи), программы и вообще любые другие информационные объекты, защищённые авторским правом.

Мошенничество

Современное мошенничество немыслимо без использования информационных технологий. На практике мошенничество часто сводится искажению финансовой документации, превышению полномочий при доступе к базе данных, модификации важной информации. В качестве защиты организациям следует использовать средства контроля над финансовой отчетностью, мониторинга всех действий пользователей и протоколирования любых операций с классифицированной информацией.

Развлечения на работе

Этот сценарий часто называют «злоупотребление сетевыми ресурсами», но на практике все сводится к одним и тем же угрозам. Во-первых, посещение сайтов неделовой (общей и развлекательной) направленности, не имеющих отношения к исполнению служебных обязанностей, в рабочее время. Во-вторых, загрузка, хранение и использование мультимедиа-файлов и развлекательных программ в рабочее время. В-третьих, использование ненормативной, грубой, некорректной лексики при ведении деловой переписки, загрузка, просмотр и распространение материалов для взрослых. В-четвертых, использование материалов, содержащих нацистскую символику, агитацию или другие противозаконные материалы. В-пятых, использование ресурсов компании для рассылки информации рекламного

характера, спама или информации личного характера, в том числе персональных данных сотрудников, финансовых данных и т.д.

Саботаж ИТ-инфраструктуры

Как уже указывалось, в экосистеме внутренних нарушителей существует чрезвычайно опасный тип инсайдеров - «обиженные» или «саботажники». Главным отличием этих злоумышленников является стремление нанести вред по личным, чаще всего бескорыстным, мотивам. Это накладывает свой отпечаток на те угрозы, которые несут саботажники, и средства, которыми они могут воспользоваться. Дело в том, что для реализации своей цели обиженные инсайдеры готовы пойти буквально на все, часто не заботясь о самосохранении.

Рейтинг опасности инсайдерских угроз

Абсолютно все аналитические отчеты указывают, что наиболее опасной инсайдерской угрозой является утечка конфиденциальной информации. Согласно исследованию «National Survey on Managing the Insider Threats», результаты которого были опубликованы Ponemon Institute в сентябре 2006 года, средний ежегодный ущерб в результате утечки информации из расчета на одну опрошенную компанию составляет 3,4 млн долл. Для сравнения аналогичный показатель потерь вследствие вирусных атак, согласно исследованию «2006 CSI/FBI Computer Crime and Security Survey», составляет менее 70 тыс долл. Кроме того, в памяти еще свежи утечки, в результате которых бизнес потерял десятки и сотни миллионов долларов. Например, компания ChoicePoint лишилась из-за кражи персональных данных почти 60 млн долл., а правительство США потратило на ликвидацию последствий утечки приватных сведений ветеранов более 500 млн. долларов.

58. Классификация и характеристика вредоносных программ, вирусов и руткитов.

У компаний-разработчиков антивирусного программного обеспечения существуют собственные классификации и номенклатуры вредоносных программ. Приведённая в этой статье классификация основана на номенклатуре «Лаборатории Касперского».

По вредоносной нагрузке

- Помехи в работе заражённого компьютера: начиная от открытия-закрытия поддона CD-ROM и заканчивая уничтожением данных и поломкой аппаратного обеспечения. Поломками известен, в частности, Win32.CIH.
  - Блокировка антивирусных сайтов, антивирусного ПО и административных функций ОС с целью усложнить лечение.
  - Саботирование промышленных процессов, управляемых компьютером (этим известен червь Stuxnet).
- Инсталляция другого вредоносного ПО.
  - Загрузка из сети (*downloader*).

    - Распаковка другой вредоносной программы, уже содержащейся внутри файла (*dropper*).
- Кража, мошенничество, вымогательство и шпионаж за пользователем. Для кражи может применяться сканирование жёсткого диска, регистрация нажатий клавиш (Keylogger) и перенаправление пользователя на поддельные сайты, в точности повторяющие исходные ресурсы.
    - Похищение данных, представляющих ценность или тайну.
    - Кража аккаунтов различных служб (электронной почты, мессенджеров,игровых серверов…). Аккаунты применяются для рассылки спама. Также через электронную почту зачастую можно заполучить пароли от других аккаунтов, а виртуальное имущество в MMOG — продать.
    - Кража аккаунтов платёжных систем.
    - Блокировка компьютера, шифрование файлов пользователя с целью шантажа и вымогательства денежных средств (см. Ransomware). В большинстве случаев после оплаты компьютер или не разблокируется, или вскоре блокируется второй раз.
    - Использование телефонного модема для совершения дорогостоящих звонков, что влечёт за собой значительные суммы в телефонных счетах.
    - Платное ПО, имитирующее, например, антивирус, но ничего полезного не делающее (fraudware или scareware (англ.)русск.; см. тж лжеантивирус).
- Прочая незаконная деятельность:
    - Получение несанкционированного (и/или дарового) доступа к ресурсам самого компьютера или третьим ресурсам, доступным через него, в том числе прямое управление компьютером (так называемыйbackdoor).
    - Организация на компьютере открытых релеев и общедоступных прокси-серверов.
    - Заражённый компьютер (в составе ботнета) может быть использован для проведения DDoS-атак.
    - Сбор адресов электронной почты и распространение спама, в том числе в составе ботнета.
    - Накрутка электронных голосований, щелчков по рекламным баннерам.
    - Генерация монет платёжной системы Bitcoin.
- Файлы, не являющиеся истинно вредоносными, но в большинстве случаев нежелательные:
    - Шуточное ПО, делающее какие-либо беспокоящие пользователя вещи.
    - Adware — программное обеспечение, показывающее рекламу.
    - Spyware — программное обеспечение, посылающее через интернет не санкционированную пользователем информацию.
    - «Отравленные» документы, дестабилизирующие ПО, открывающее их (например, архив размером меньше мегабайта может содержать гигабайты данных и надолго «завесить» архиватор).

- Программы удалённого администрирования могут применяться как для того, чтобы дистанционно решать проблемы с компьютером, так и для неблаговидных целей.
- Руткит нужен, чтобы скрывать другое вредоносное ПО от посторонних глаз.
- Иногда вредоносное ПО для собственного «жизнеобеспечения» устанавливает дополнительные утилиты: IRC-клиенты[5], программныемаршрутизаторы[6], открытые библиотеки перехвата клавиатуры…[7]Такое ПО вредоносным не является, но из-за того, что за ним часто стоит истинно вредоносная программа, детектируется антивирусами. Бывает даже, что вредоносным является только скрипт из одной строчки, а остальные программы вполне легитимны.[8]

По методу размножения

- Эксплойт — теоретически безобидный набор данных (например,графический файл или сетевой пакет), некорректно воспринимаемый программой, работающей с такими данными. Здесь вред наносит не сам файл, а неадекватное поведение ПО с ошибкой. Также эксплойтом называют программу для генерации подобных «отравленных» данных.

- Логическая бомба в программе срабатывает при определённом условии, и неотделима от полезной программы-носителя.
- Троянская программа не имеет собственного механизма размножения.
- Компьютерный вирус размножается в пределах компьютера и через сменные диски. Размножение через сеть возможно, если пользователь сам выложит заражённый файл в сеть. Вирусы, в свою очередь, делятся по типу заражаемых файлов (файловые, загрузочные, макро-, автозапускающиеся); по способу прикрепления к файлам (паразитирующие, «спутники» и перезаписывающие) и т. д.
- Сетевой червь способен самостоятельно размножаться по сети. Делятся на IRC-, почтовые, размножающиеся с помощью эксплойтов и т. д.

Вредоносное ПО может образовывать цепочки: например, с помощью эксплойта (1) на компьютере жертвы развёртывается загрузчик (2), устанавливающий из интернета червя (3).

Классификация руткитов

- **По уровню привилегий**
  - Уровень пользователя (*user-mode*)
  - Уровень ядра (*kernel-mode*)
- **По принципу действия**
  - изменяющие алгоритмы выполнения системных функций (*Modify execution path*)
  - изменяющие системные структуры данных (*Direct kernel object manipulation*)[1]

59. Методы антивирусной защиты.

Говоря о системах Майкрософт, следует знать, что обычно антивирус действует по схеме:

- поиск в базе данных антивирусного ПО сигнатур вирусов.
- если найден инфицированный код в памяти (оперативной и/или постоянной), запускается процесс «карантина», и процесс блокируется.
- зарегистрированная программа обычно удаляет вирус, незарегистрированная просит регистрации и оставляет систему уязвимой.

Сегодня используется несколько основополагающих методик обнаружения и защиты от вирусов:

- сканирование;
- эвристический анализ;
- использование антивирусных мониторов;
- обнаружение изменений;
- использование антивирусов, встроенных в BIOS компьютера.

Кроме того, практически все антивирусные программы обеспечивают автоматическое восстановление зараженных программ и загрузочных секторов. Конечно, если это возможно.

60. Брандмауэры и антивирусные программы.

Термин **брандмауэр** или его английский эквивалент **файрвол**(англ. *firewall*) используется также в значении «**межсетевой экран**»

Межсетево́й экра́н,сетево́й экра́н — это комплекс аппаратных и программных средств в компьютерной сети, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Антиви́русная програ́мма (антиви́рус) — специализированная программа для обнаружения компьютерных вирусов, а также нежелательных (считающихся вредоносными) программ вообще и восстановления заражённых (модифицированных) такими программами файлов, а также для профилактики — предотвращения заражения (модификации) файлов или операционной системы вредоносным кодом.

61. Классическая архитектура ОС (на основе ядра), состав и функции ядра и вспомогательных модулей ОС.

Функциональная сложность ОС неизбежно приводит к сложности ее **архитектуры**, под которой обычно понимается:

- структурная организация ОС на основе различных программных модулей;
- логическая организация системы с точки зрения ее пользователей.

ОС					
Ядро		**Вспомогательные модули ОС**			
Модули, решающие внутрисистемные задачи	Модули, обеспечивающие поддержку приложений	Утилиты	Системные обрабатывающие программы	Сервисные программы	Библиотеки процедур различного назначения

Модули ядра выполняют такие базовые функции ОС, как управление процессами, памятью, устройствами ввода-вывода (УВВ) и другие. Ядро составляет сердцевину, без которой ОС становится полностью неработоспособной и не сможет выполнить ни одной своей функции. В состав ядра входят:

- модули, решающие внутрисистемные задачи организации вычислительного процесса (переключение контекстов, загрузка-выгрузка страниц, обработка прерываний). Эти функции недоступны для приложений;
- модули, обеспечивающие поддержку приложений (создание прикладной программной среды). Приложения могут обращаться к ядру с запросами – системными вызовами – для выполнения тех или иных действий, например, открытия и чтения файла, вывода графической информации, получения системного времени и других. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования (Application Programming Interface, API).

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями ОС. Поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в ОП.

Вспомогательные модули ОС обычно подразделяются на следующие группы:
- утилиты – программы, решающие отдельные задачи управления и сопровождения компьютерной системы, например, программы сжатия дисков, архивирования данных;
- системные обрабатывающие программы – компиляторы, компоновщики, отладчики, профайлеры и т.п.;
- сервисные программы, предоставляющие пользователю дополнительные услуги, такие как специальный вариант графического интерфейса пользователя (ГИП), калькулятор, игры и т.д.;
- библиотеки процедур различного назначения, упрощающие разработку приложений, например, библиотека математических функций, библиотека функций ввода-вывода и т.д.

62. Режимы и средства обеспечения привилегий ОС.

Многоуровневая иерархия привилегий.

Средства аппаратной поддержки ОС					
1. Средства поддержки привилегиро-ванного режима	2. Средства трансляции адресов	3. Средства переключения процессов	4. Система прерываний	5. Системный таймер	6. Средства защиты областей памяти

Система прерываний – позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу УВВ, быстро переключаться с одной программы на другую.

Системный таймер – быстродействующий регистр-счетчик, обеспечивающий выдержку необходимых ОС интервалов времени.

Средства защиты областей памяти – обеспечивают проверку возможности чтения, записи или выполнения кода данной области ОП.

Средства переключения процессов – предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным.

Средства поддержки привилегированного режима – обеспечивают смену режима работы процессора.

Средства трансляции адресов – преобразуют виртуальные адреса в кодах процесса в физические.

Обеспечить привилегии ОС нельзя без специальных *средств аппаратной поддержки*. Аппаратура компьютера должна поддерживать как минимум два режима работы – **пользовательский режим** (user mode) и **привилегированный режим** или так называемый **режим ядра** (kernel mode) или режим супервизора (supervisor mode). Тогда ОС или ее части (и в первую очередь, ядро) работают в привилегированном режиме, а все приложения – в пользовательском режиме.

В архитектуре x86 существует 4 уровня привилегий или кольца. В процессоре без аппаратной поддержки виртуализации в кольце 0, имеющем максимальные привилегии, выполняется ядро ОС, имея полный доступ к процессору. Выполнение инструкций в кольце 0 соответствует привилегированному режиму. Кольца 1 и 2 не используются, а в кольце 3 работают приложения. Выполнение инструкций в кольце 3 соответствует пользовательскому режиму.

Число уровней привилегий, поддерживаемых ОС на базе реализуемых аппаратно, для разных систем различно. Например, на базе 4 уровней для

процессоров Intel: в ОС OS/2 построена трехуровневая система привилегий, а в ОС семейств Windows, UNIX и некоторых других – двухуровневая. То есть ОС может обеспечить сколь угодно развитую программную систему защиты.

Эта система защиты может поддерживать, например, многоуровневую иерархию привилегий, что позволяет более тонко распределять полномочия, как между модулями ОС, так и между приложениями. Появление внутри ОС более и менее привилегированных частей позволяет повысить ее устойчивость к внутренним ошибкам программных кодов, так как эти ошибки будут касаться только модулей определенного уровня привилегий. Разграничение привилегий в среде приложений позволяет строить сложные прикладные ПС, в которых часть более привилегированных модулей может получать доступ к данным менее привилегированных и управлять их выполнением.

63. Многослойная структура ОС, особенности слоев и межслойных интерфейсов.

Вычислительную систему (ВС), работающую под управлением ОС на основе монолитного ядра, можно представить в виде трехуровневой иерархии слоев: нижний – аппаратура, средний – ядро, верхний – вспомогательные модули ОС (рис.3.2).

Особенностью этой иерархии является то, что отдельный слой может взаимодействовать только со смежными слоями. Например, приложения не могут сами взаимодействовать с аппаратурой.

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе и ПС. В соответствии с ним:

- система состоит из иерархии слоев (рис.3.3);
- каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций межслойного интерфейса;

На основе функций нижележащего слоя следующий, вышележащий слой строит свои функции – более сложные и более мощные. Они являются основой для создания еще более мощных функций следующего вышележащего слоя;

- строгие правила касаются только взаимодействия между слоями системы, а между модулями внутри слоя связи могут быть произвольными;
- отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться за помощью к нижележащему слою через межслойный интерфейс.

Такая организация системы имеет много достоинств. Она существенно упрощает разработку ОС, так как позволяет сначала определить «сверху вниз» функции слоев и межслойные интерфейсы, а затем при детальной реализации постепенно наращивать мощность функций слоев, двигаясь «снизу вверх». Кроме того, при модернизации ОС можно изменять модули внутри слоев без необходимости изменений в остальных слоях, если при таких внутренних изменениях остаются в силе межслойные интерфейсы.

64. Типовые слои ядра ОС и их функции.

5. Интерфейс системных вызовов
4. Менеджеры ресурсов
3. Базовые механизмы ядра
2. Машинно-зависимые компоненты ОС
1. Средства аппаратной поддержки ОС

1. *Средства аппаратной поддержки ОС*. Часть функций ОС может выполняться аппаратными средствами, непосредственно участвующими в организации вычислительных процессов, такими как: средства поддержки привилегированного режима, система прерываний, средства переключения контекстов процессов, средства защиты областей памяти и другие.
2. Машинно-зависимые компоненты ОС – это программные модули, отражающие специфику аппаратной платформы компьютера. В идеале этот слой должен изолировать вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои из машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС. Например, машинно-зависимые компоненты ОС содержит слой аппаратной абстракции (Hardware Abstraction Layer, HAL) в ОС Windows NT.

3. *Базовые механизмы ядра*. Этот слой выполняет наиболее примитивные операции ядра, такие как: программное переключение контекстов процессов, диспетчеризация прерываний, перемещение страниц из ОП на диск и обратно и т.п. Модули данного слоя не принимают решений о распределении ресурсов, они только отрабатывают принятые «наверху» решения, почему и называются исполнительными механизмами для модулей верхних слоев. Например, решение о том, что в данный момент нужно прервать выполнение текущего процесса А и начать выполнение процесса В, принимается менеджером процессов в вышележащем слое (4), а слою базовых механизмов (3) передается только директива, что нужно выполнить переключение с контекста текущего процесса А на контекст нового процесса В;

4. *Менеджеры ресурсов*. Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами ВС. Обычно в этом слое работают менеджеры (диспетчеры) процессов, ввода-вывода, файловой системы (ФС) и ОП. Разбиение на менеджеры может быть и несколько иным. Например, менеджер ФС иногда объединяют с менеджером ввода-вывода, а функции управления доступом пользователей к системе в целом и ее отдельным объектам поручают отдельному менеджеру безопасности. Каждый менеджер ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение по запросам приложений.

5. *Интерфейс системных вызовов*. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя API ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Например, в UNIX с помощью системного вызова fd = open (“/doc/a.txt”, O_RDONLY) приложение открывает файл a.txt, хранящийся в каталоге /doc, а с помощью системного вызова read (fd, buffer, count) читает из этого файла в область своего адресного пространства, имеющую имя buffer, некоторое число байтов. Для осуществления таких комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов (4), причем для выполнения одного системного вызова может понадобиться несколько таких обращений.

65. Менеджеры ресурсов.

Этот слой состоит из функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами ВС. Обычно на данном слое работают менеджеры (диспетчеры) процессов, ввода-вывода, файловой системы и оперативной памяти. Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений. Для исполнения принятых решений менеджер обращается к слою базовых механизмов с запросами. Внутри слоя менеджеров существуют тесные взаимные связи, так как для выполнения процессу нужен доступ одновременно к нескольким ресурсам - процессору, области памяти, возможно, к определенному файлу или устройству ввода-вывода. Например, при создании процесса менеджер процессов обращается к менеджеру памяти, который должен выделить процессу определенную область памяти.

66. Типовые средства аппаратной поддержки ОС.

Четкая граница между программной и аппаратной реализацией функций ОС отсутствует. Подобные решения принимают разработчики аппаратного и программного обеспечения компьютера.

Но практически все современные аппаратные платформы имеют некоторый типовой набор средств аппаратной поддержки ОС, в который входят следующие 6 средств:

1. Средства поддержки привилегированного режима

2. Средства трансляции адресов

3. Средства переключения процессов

4. Система прерываний

5. Системный таймер

6. Средства защиты областей памяти

Средства поддержки привилегированного режима обеспечивают смену режима работы процессора. Основой для них является системный регистр процессора – так называемое «слово состояния процессора» (ССП) или «слово состояния машины» (ССМ). Этот регистр содержит некоторые признаки режимов работы процессора, в том числе признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения ССП в результате прерывания или выполнения привилегированной команды. Число градаций привилегированности у разных типов процессоров различное. Наиболее часто используются 2 уровня (ядро – пользователь) или 4 уровня (ядро – супервизор – выполнение – пользователь у процессоров VAX или 1-2-3-4 у процессоров Intel x86/Pentium). При этом обязательна проверка допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности. 2. Средства трансляции адресов преобразуют виртуальные адреса в кодах процесса в адреса физической ОП. Для трансляции используются таблицы большого объема в ОП, а аппаратура процессора содержит только указатели на эти области ОП. Средства трансляции адресов используют эти указатели для доступа к элементам таблиц при более быстром аппаратном выполнении алгоритма преобразования адреса. 3. Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным. Содержимое контекста обычно включает содержимое всех регистров общего назначения процессора, регистра флагов операций (нуля, переноса, переполнения и т.п.), а также тех системных указателей, которые связаны с отдельным процессом, а не с самой ОС, например, указателя на таблицу 18 трансляции адресов процесса. Для хранения контекстов приостанавливаемых процессов обычно используются области ОП, поддерживаемые указателями процессора. Переключение контекста выполняется по определенным командам процессора, например, по команде перехода на новую задачу. Такая команда автоматически вызывает загрузку данных из сохраненного

контекста в регистры процессора, после чего процесс продолжается с ранее прерванного места. 4. Система прерываний позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу УВВ, быстро переключаться с одной программы на другую. Механизм прерываний нужен для того, чтобы оповестить процессор о возникновении в ВС некоторого непредсказуемого или не синхронизированного с циклом работы процессора события, для чего источник прерывания вырабатывает определенный электрический сигнал. Известный переход на процедуру обработки прерывания сопровождается заменой ССП (или даже всего контекста процесса), что позволяет перейти в привилегированный режим. Даже системные вызовы от приложений выполняются на многих аппаратных платформах с помощью специальной инструкции прерывания, вызывающей переход к выполнению соответствующих процедур ядра. 5. Системный таймер реализуется в виде быстродействующего регистра-счетчика и необходим ОС для выдержки интервалов времени. Для этого в регистр таймера программно загружается значение интервала в условных единицах, из которого затем с определенной частотой начинает вычитаться по единице. При достижении нулевого значения таймер инициирует прерывание. Прерывания от системного таймера используются ОС прежде всего для слежения за тем, как отдельные процессы расходуют время процессора. 6. Средства защиты областей памяти обеспечивают проверку возможности чтения, записи или выполнения кода данной области ОП. Эти средства обычно встраиваются в механизм трансляции адресов. Аппаратные функции защиты памяти состоят в сравнении уровней привилегий текущего кода процессора и сегмента памяти, к которому производится обращение.

67.Особенности построения машинно-зависимых компонент и переносимости ОС.

Машинно-зависимые компоненты ОС – это программные модули, отражающие специфику аппаратной платформы компьютера. В идеале этот слой должен изолировать вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои из машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС. Например, машинно-зависимые компоненты ОС содержит слой аппаратной абстракции (Hardware Abstraction Layer, HAL) в ОС Windows NT.

Многие ОС успешно работают на различных аппаратных платформах без существенных изменений в своем составе. Несмотря на различия в деталях, средства аппаратной поддержки ОС большинства компьютеров приобрели сегодня множество типовых черт, серьезно влияющих на работу компонентов ОС. В результате в ОС можно выделить достаточно компактный слой машинно-зависимых компонентов ядра и сделать остальные слои общими для разных аппаратных платформ.

68. Микроядро: понятие и состав. Серверы ОС.

Микроядро реализует базовые функции операционной системы, на которые опираются другие системные службы и приложения. Основной проблемой при конструировании микроядерной ОС является распознавание тех функций системы, которые могут быть вынесены из ядра. Такие важные компоненты ОС как файловые системы, системы управления окнами и службы безопасности становятся периферийными модулями, взаимодействующими с ядром и друг с другом.

Когда-то казалось, что многоуровневая архитектура ядра ОС UNIX является вершиной в области конструирования операционных систем. Основные функциональные компоненты операционной системы - файловая система, взаимодействие процессов (IPC - interprocess communications), ввод-вывод и управление устройствами - были разделены на уровни, каждый из которых мог взаимодействовать только с непосредственно примыкающим к нему уровнем.

В микроядерных архитектурах вертикальное распределение функций операционной системы заменяется на горизонтальное. Компоненты, лежащие выше микроядра, используют средства микроядра для обмена сообщениями, но взаимодействуют непосредственно. Микроядро лишь проверяет законность сообщений, пересылает их между компонентами и обеспечивает доступ к аппаратуре.

Совсем иная ситуация возникает в том случае, когда в виде приложения оформляется часть ОС. По определению, основным назначением такого приложения является обслуживание запросов других приложений. Это касается, например, порождения процесса, выделения ему памяти, проверки его прав доступа к ресурсу и т.п. Именно поэтому менеджеры ресурсов, вынесенные в пользовательский режим, и стали называться серверами ОС – модулями, обслуживающими запросы приложений и других модулей ОС. Значит, для реализации архитектуры на основе микроядра необходимо обеспечить удобный и эффективный способ вызова процедур одного процесса из другого, причем поддержка такого механизма как раз и является одной из главных задач микроядра.

69.Архитектура ОС на основе микроядра. Механизм обращения к функциям ОС.

Архитектура на основе микроядра является альтернативой рассмотренному классическому способу построения ОС, когда все функции ядра выполняются в привилегированном режиме. Ее суть состоит в том, что в привилегированном режиме остается работать только очень небольшая часть ядра, называемая микроядром и защищенная от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие часть базовых функций ядра по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений

и управлению УВВ, связанные с загрузкой или чтением регистров устройств. Набор базовых функций микроядра обычно соответствует функциям слоя базовых механизмов обычного ядра. Такие функции ОС трудно или невозможно выполнить в пространстве пользователя. А все остальные более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме. Однако однозначного решения о способе подобного разделения ядра не существует. В общем случае многие менеджеры ресурсов (такие как ФС, подсистемы управления виртуальной памятью и процессами, менеджер безопасности и т.п.) становятся «периферийными» модулями, работающими в пользовательском режиме. Но работающие в пользовательском режиме менеджеры ресурсов принципиально отличаются от традиционных утилит и обрабатывающих программ, хотя и также оформлены в виде приложений. Утилиты и обрабатывающие программы обычно вызываются только пользователями. Ситуации же, когда одному приложению требуется выполнение функции (или процедуры) другого приложения, возникают крайне редко. Именно поэтому в ОС с классической архитектурой отсутствует механизм вызова одним приложением функций другого.

В ОС на основе микроядра механизм обращения к функциям ОС, оформленным в виде серверов, основан на модели взаимодействия «клиент-сервер» (рис.3.6). Клиент (например, приложение или компонент ОС) запрашивает выполнение некоторой функции сервера, посылая сообщение (обозначим его цифрой 1) микроядру. Отметим, что непосредственная передача сообщений между приложениями невозможна, так как их адресные пространства изолированы. Но микроядро, выполняющееся в привилегированном режиме, имеет доступ к адресным пространствам приложений и поэтому может стать посредником, передающим сообщения: нужному серверу – имя и параметры вызываемой процедуры (2), обратно клиенту – результаты (3, 4). В свою очередь, серверы через микроядро могут подобным образом включать в работу друг друга, как показано на рис.3.6 соответственно по шагам цифрами 5-8. Отметим, что в рассмотренных примерах смена режимов происходит 4 раза (1→2→3→4; 5→6→7→8).

70. Новые варианты ядра и архитектуры ОС.

1. Монолитное ядро

2. Микроядро

3. Гибридное ядро

4. Модульное ядро

5. Наноядро

7. Пикоядро

6. Экзоядро

1. *Монолитное ядро* является наиболее архитектурно зрелым и пригодным к эксплуатации, оно представляет собой набор процедур, работающих в привилегированном режиме. Каждая процедура может вызвать любую другую. Монолитное ядро предоставляет богатый набор абстракций оборудования, все его компоненты работают в одном адресном пространстве, являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путем непосредственного вызова процедур. Достоинства монолитного ядра: скорость работы, упрощенная разработка модулей, богатство предоставляемых возможностей и функций, поддержка большого количества разнообразного оборудования. Недостатки: поскольку все компоненты ядра работают в одном адресном пространстве, сбой в одном из них может нарушить работоспособность всей системы; монолитность ядра усложняет отладку и понимание его кода, добавление новых функций и возможностей, удаление устаревшего кода; избыточность кода монолитного ядра повышает затраты ОП для его функционирования.

2. *Микроядро* представляет лишь необходимую часть монолитного ядра. Классическое микроядро предоставляет очень небольшой набор низкоуровневых примитивов, или системных вызовов, реализующих базовые сервисы ОС, в число которых входят: управление адресным пространством ОП и виртуальной памяти, процессами и потоками, а также средства межпроцессного взаимодействия. Такая конструкция позволяет улучшить общее быстродействие системы, так как небольшое микроядро может уместиться в кэше процессора, но вызывает избыточные смены режимов работы процессора. Поскольку рассмотренные подходы к построению ОС имеют свои достоинства и недостатки, в большинстве случаев современные и перспективные ОС используют различные модификации или комбинации этих основных подходов.

3. *Гибридное (смешанное) ядро* представляет собой модифицированное микроядро, разрешающее для ускорения работы запускать «несущественные» части в пространстве ядра. Например, ОС Linux имеет монолитное ядро с элементами микроядра. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра (модулей). В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться экспортируемые ядром функции. Существуют варианты ОС GNU, в которых вместо монолитного ядра применяется ядро Mach, а поверх него функционируют в пользовательском режиме те же процессы, которые при использовании Linux были бы частью ядра. Другим примером такого смешанного подхода является возможность запуска ОС

с монолитным ядром под управлением микроядра. Например, так устроены ОС 4.4BSD и MkLinux, основанные 27 на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. В то время как все остальные функции, в том числе взаимодействие с прикладными программами, осуществляется монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества архитектуры на основе микроядра, сохраняя по возможности хорошо отлаженный код монолитного ядра. Но все же наиболее тесно элементы монолитного ядра и микроядра переплетены в ОС Windows NT. Ее микроядро NT kernel занимает более 1 Мбайт и слишком велико для идеального микроядра. Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в ОС на основе микроядра. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно именно ОС с монолитным ядром. Поэтому Windows NT можно с полным правом назвать гибридной ОС. NT kernel использовано в Windows NT и последующих. Другие примеры гибридных ядер: ядро ОС BeOS, микроядро DragonFly BSD, ядро сетевой ОС NetWare.

4. *Модульное ядро* представляет собой современную, усовершенствованную модификацию монолитного ядра. Модульное ядро, как правило, уже не требует полной его перекомпиляции при изменении состава аппаратного обеспечения компьютера. Модульное ядро поддерживает механизм загрузки поддерживающих аппаратуру модулей ядра (например, драйверов). При этом загрузка модулей может быть динамической без перезагрузки ОС или статической, выполняемой при перезагрузке ОС после реконфигурирования системы. Модульное ядро удобнее разрабатывать, чем традиционные монолитные ядра, не поддерживающие динамическую загрузку модулей, так как от разработчика не требуется многократная полная перекомпиляция ядра при работе над какой-либо его подсистемой или драйвером. Выявление, локализация, отладка и устранение ошибок при тестировании также облегчаются. 28 Модульное ядро предоставляет особый API для связывания модулей с ядром, для обеспечения динамической загрузки и выгрузки модулей. Модули ядра вместо функций стандартной библиотеки C/C++ должны использовать специальные функции API ядра, а также обязаны экспортировать определённые функции, нужные ядру для правильного подключения и распознавания модуля, для его корректной инициализации при загрузке и корректного завершения при выгрузке, для регистрации модуля в таблице модулей ядра и для обращения из ядра к сервисам, предоставляемым модулем. Но, к сожалению, не все части ядра могут быть сделаны модулями. Некоторые части ядра всегда обязаны присутствовать в ОП и должны быть жёстко «вшиты» в ядро. Также не все модули допускают динамическую загрузку. Степень модульности ядра (оцениваемая как количество и разнообразие кода,

которое может быть вынесено в отдельные модули ядра и допускает динамическую загрузку) различна в различных реализациях модульного ядра. Например, ОС Linux в настоящее время имеет модульную архитектуру, но модульность «нестабильна», поскольку API меняется от релиза к релизу, в отличие от стабильных интерфейсов модульных ядер *BSD (FreeBSD, NetBSD, OpenBSD). Общей тенденцией развития является повышение степени модульности ядра, улучшение механизмов динамической загрузки и выгрузки модулей, уменьшение или устранение необходимости в ручной загрузке модулей или в реконфигурации ядра при изменениях аппаратуры путём введения тех или иных механизмов автоматического определения оборудования и автоматической загрузки нужных модулей, универсализация кода ядра и введение в ядро абстрактных механизмов, предназначенных для совместного использования многими модулями Примером может служить виртуальная файловая система VFS, совместно используемая многими модулями файловых систем в ОС Linux.

5. *Наноядро* представляет собой крайне упрощённое и минималистичное ядро, выполняющее лишь одну задачу – обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки аппаратных прерываний такое наноядро посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему ПО при помощи того же механизма прерываний. При этом предоставляются удобные механизмы абстракции от конкретных устройств и способов обработки их прерываний. В современных компьютерах наноядро часто используется для виртуализации аппаратного обеспечения реальных компьютеров или с целью позволить нескольким или многим различным ОС работать одновременно и параллельно на одном компьютере. Например, среда VMware ESX Server реализует собственное наноядро, независимое от ОС и устанавливаемое вообще без ОС. Под его управлением работают пользовательские и административные утилиты VMware, а также необходимые ОС, виртуализируемые в среде ESX Server. Наноядро также может использоваться для обеспечения переносимости (портируемости) ОС на разные аппаратные платформы или для обеспечения возможности запуска некоторой «старой» ОС на новой, несовместимой аппаратной платформе без ее полного переписывания и портирования. Например, компания Apple использовала наноядро в версии ОС Mac OS Classic на платформе PowerPC для того, чтобы транслировать аппаратные прерывания, генерировавшиеся компьютерами на базе процессоров PowerPC в форму, которая могла бы «пониматься» и распознаваться Mac OS для процессоров Motorola 680x0. Таким образом, наноядро эмулировало для Mac OS «старую» 680x0 аппаратуру. Альтернативой было бы полное переписывание и портирование кода Mac OS на PowerPC при переходе с 680x0 на них. Позднее, в эпоху Mac OS 8.6, наноядро виртуализировало предоставляемые PowerPC мультипроцессорные возможности и обеспечивало поддержку симметричного мультипроцессирования (Symmetrical Multiprocessing, SMP) в Mac

OS. Другим удачным примером является наноядро Adeos, работающее как модуль ядра Linux и позволяюшее выполнять одновременно с Linux какую-либо ОС реального времени (РВ). Заметим также, что термин «наноядро» иногда неформально используется для описания очень маленьких, упрощённых и лёгких микроядер.

6. *Пикоядро*. Наноядро может быть настолько маленьким и примитивным, что даже важнейшие устройства, находящиеся непосредственно на материнской плате или на плате контроллера встраиваемого устройства (такого как таймер или программируемый контроллер прерываний) обслуживаются уже не им, а специальными драйверами устройств. Такого рода сверхминималистичное наноядро называется пикоядром.

7. *Экзоядро* (экзо – приставка, обозначающая нечто внешнее, находящееся снаружи) предоставляет лишь функции для взаимодействия процессов и безопасного выделения и освобождения ресурсов. В отличие от монолитного ядра, предоставляющего не только минимальный набор сервисов выполнения программ, но и большое число высокоуровневых абстракций для использования разнородных ресурсов компьютера (ОП, жестких дисков, сетевых подключений), экзоядро предоставляет лишь набор сервисов для взаимодействия процессов, а также минимальные функции защиты: выделение и высвобождение ресурсов, контроль прав доступа, и т. д. Предоставление абстракций для физических ресурсов выносится в библиотеку пользовательского уровня libOS. libOS может обеспечивать произвольный набор абстракций, совместимый с той или иной уже существующей ОС, например, Linux или Windows.

71. Суть и виды совместимости различных ОС, трансляция библиотек.

В результате предшествующего рассмотрения вопросов совместимости различных ОС становится ясно, что создание полноценной ППС(прикладная программная среда), полностью совместимой со средой другой ОС, является достаточно сложной задачей, тесно связанной со структурой ОС. В то же время известны различные подходы и варианты построения множественных ППС, различающиеся архитектурными особенностями, функциональными возможностями, различной степенью переносимости приложений. Например, в UNIX транслятор ППС реализуется в виде обычного приложения. В ОС с микроядром (Windows NT и последующих) ППС выполняются в виде серверов пользовательского режима. В OS/2 (с ее более простой архитектурой) средства организации прикладных программных сред встроены глубоко в ОС.

Выходом в таких случаях является использование прикладных программных сред. Одной из составляющих, формирующих прикладную

программную среду, является набор функций интерфейса прикладного программирования API, которые ОС предоставляет своим приложениям. Для сокращения времени на выполнение чужих программ прикладные среды имитируют обращение к библиотечным функциям.

Эффективность этого подхода связана с тем, что большинство сегодняшних программ работают под управлением GUI (графический интерфейс пользователя) типа Windows, Mac или UNIX Motif, поэтому приложения тратят большую часть времени, производя некоторые хорошо предсказуемые действия. Они непрерывно выполняют вызовы библиотек GUI для манипулирования окнами и для других связанных с GUI действий. Сегодня в типичных программах 60 – 80 % времени тратятся на выполнение функций GUI и других библиотечных вызовов ОС. Именно это свойство приложений позволяет прикладным средам компенсировать большие затраты времени, потраченные на покомандное эмулирование программы. Тщательно спроектированная программная среда имеет в своем составе библиотеки, имитирующие внутренние библиотеки GUI, но написанные на “родном” коде. Таким образом, достигается существенное ускорение выполнения программ с API другой ОС. Иногда такой подход называют трансляцией для того, чтобы отличать его от более медленного процесса эмулирования кода по одной команде за раз.

Чтобы программа, написанная для одной ОС, могла быть вызвана в рамках другой ОС, не достаточно лишь обеспечить совместность API. Концепции, положенные в основу разных ОС, могут входить в противоречие друг с другом. Например, в одной ОС приложению может быть разрешено непосредственно управлять устройством ввода-вывода, в другой – эти действия являются прерогативой ОС. Каждая ОС имеет свои собственные механизмы защиты ресурсов, свои алгоритмы обработки ошибок и исключительных ситуаций, особую структуру процесса и схему управления памятью, свою семантику доступа к файлам и графический пользовательский интерфейс.

Для обеспечения совместимости необходимо организовать бесконфликтное сосуществование в рамках одной ОС нескольких способов управления ресурсами компьютера.

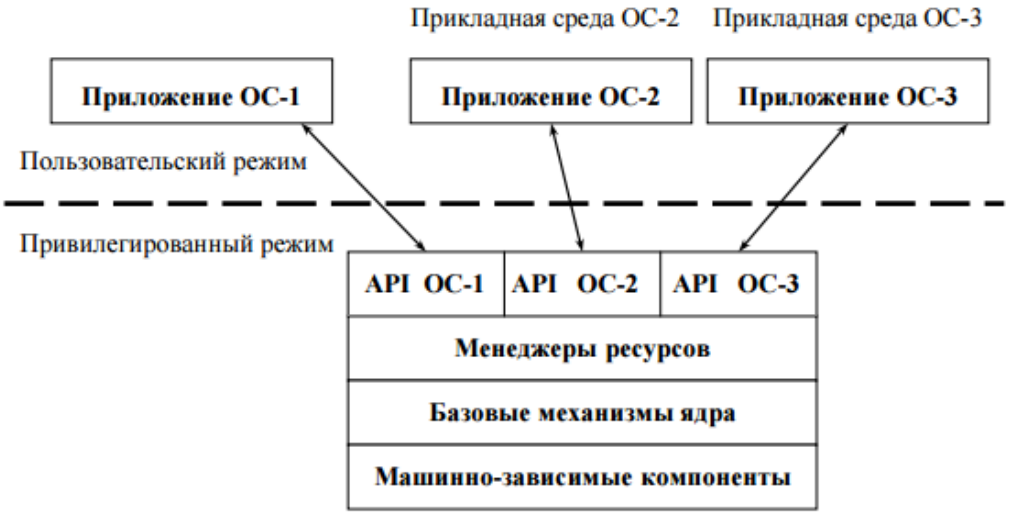
72. Вариант реализации множественных прикладных программных сред на основе трансляторов системных вызовов.

Во втором варианте реализации множественных ППС основная ОС имеет несколько равноправных API, размещенных непосредственно в пространстве ее ядра. Здесь функции уровня API обращаются к функциям нижележащего уровня ОС, которые должны поддерживать все объединяемые ППС. В то же время известно, что в разных ОС по-разному осуществляется управление системным временем, используется разный формат времени дня, по-своему разделяется процессорное время и т.д. Привилегированный режим Транслятор системных

вызовов Пользовательский режим API ОС-1 Прикладная среда ОС-2 Прикладная среда ОС-3 Менеджеры ресурсов Базовые механизмы ядра Машинно-зависимые компоненты Транслятор системных вызовов Приложение ОС-2 (API ОС-2) Приложение ОС-3 (API ОС-3) Приложение ОС-1 35 Поэтому функции каждого API в этом варианте реализуются ядром с учетом специфики (и по правилам) его ОС. Например, таковы функции создания и завершения процессов. Третий вариант реализации множественных ППС основан на концепции микроядра.

73. Вариант реализации множественных прикладных программных сред на основе поддержки нескольких равноправных API.

Улучшить ситуацию в вопросе совместимости может использование нескольких ППС, основу которых составляет набор функций API. Для ускорения выполнения чужих программ ППС имитируют обращения к чужим библиотечным функциям. Эффективность такого подхода связана с тем, что большинство приложений работает под управлением ГИП типа Windows или Motif с большими затратами времени (60-80%) на хорошо предсказуемые действия – непрерывные вызовы библиотек ГИП для манипулирования окнами, пиктограммами и т.д. Эта особенность приложений позволяет ППС компенсировать большие затраты времени на покомандное эмулирование программы. При этом качественная ППС включает библиотеки, имитирующие внутренние библиотеки ГИП, но написанные на «родном» языке, что существенно ускоряет выполнение программ с API другой ОС. Такой подход называют трансляцией, чтобы отличить от медленного покомандного эмулирования. Но чтобы выполнить программу одной ОС в среде другой, совместимости API оказывается недостаточно. Дело здесь в том, что каждая ОС обычно имеет собственные механизмы ввода-вывода, защиты ресурсов, алгоритмы обработки ошибок и исключительных ситуаций, особую структуру процесса и схему управления памятью, свою семантику доступа к файлам и ГИП. Таким образом, для обеспечения совместимости необходимо организовать бесконфликтное сосуществование в рамках одной ОС нескольких способов управления ресурсами компьютера



74. Вариант реализации множественных прикладных программных сред на основе концепции микроядра.

Вариант реализации множественных ППС основан на концепции микроядра.

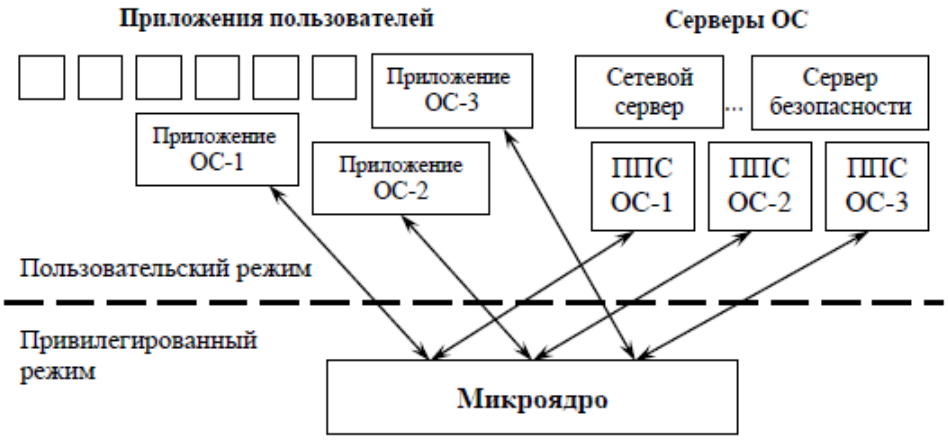


Рис. 3.10. Третий вариант реализации множественных ППС на основе концепции микроядра

При этом очень важно отделить базовые, общие для всех ППС, механизмы ОС от специфических для каждой ППС высокоуровневых функций, решающих стратегические задачи. Здесь все функции ОС реализуются микроядром и серверами пользовательского режима. Важно, что каждая ППС оформляется в виде отдельного сервера пользовательского режима и не включает базовых механизмов ОС. А приложения, используя API, обращаются через микроядро с системными вызовами к соответствующей ППС. ППС обрабатывает запрос, выполняет его и отсылает результат ожидающему его приложению. Заметим, что в ходе выполнения запроса ППС приходится, в свою очередь, обращаться за помощью к базовым механизмам ОС, реализуемым микроядром, и другим серверам ОС.

Последнему подходу к конструированию множественных ППС присущи все достоинства и недостатки архитектуры на основе микроядра:

- очень просто можно добавлять и исключать ППС, что является следствием хорошей расширяемости ОС на основе микроядра;
- надежность и стабильность выражаются в том, что при отказе одной ППС все остальные сохраняют работоспособность;
- низкая производительность ОС на основе микроядра сказывается на скорости работы ППС, а значит и на скорости выполнения приложений.

Таким образом, создание в рамках одной ОС нескольких ППС для выполнения приложений разных ОС позволяет иметь единственную версию программы и переносить ее между разными ОС. Заметим, что множественные ППС обеспечивают совместимость на двоичном уровне данной ОС с приложениями, написанными для других ОС. В результате пользователи получают большую свободу выбора ОС и более легкий доступ к качественному ПО.

75. Гипервизоры.

Гипервизор – перспективная компьютерная программа или аппаратная схема, обеспечивающая совместное (одновременное, параллельное) выполнение двух и более немодифицированных ОС на одном компьютере. Гипервизор обеспечивает изоляцию работающих ОС друг от друга, защиту и безопасность, разделение ресурсов между различными запущенными ОС и управление ресурсами. Гипервизор уже сегодня претендует на роль базового технологического уровня ПО компьютера, который действует как основа для гостевых ОС.

Гипервизор также может предоставлять работающим под его управлением ОС средства связи и взаимодействия (например, обмен файлами или сетевые соединения) так, как если бы эти ОС выполнялись на разных физических компьютерах. Таким образом, гипервизор, по сути, является минимальной ОС с микроядром или наноядром. Он предоставляет управляемым ОС **сервис ВМ**, при необходимости эмулируя реальную аппаратуру, управляет организованными ВМ, распределяет ресурсы для них.

При этом гипервизор обеспечивает независимое «включение» и «выключение», а также перезагрузку любой виртуальной машины с необходимой пользователю ОС.

Новый термин «гипервизор» введен специально, чтобы подчеркнуть его отличие от термина «супервизор», которым традиционно называли ядро ОС, а точнее, менеджер ресурсов ядра в эпоху мейнфреймов. Гипервизор является расширением и обобщением понятия «супервизор». Как супервизор в ядре ОС обеспечивал изоляцию приложений друг от друга, выделение и освобождение ресурсов для пользовательских процессов, так и гипервизор обеспечивает изоляцию и управление ресурсами, только уже для целых ОС со всеми их особенностями и различиями.

76. Технологии виртуализации.

Виртуализация, в широком смысле, это предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию вычислительных процессов, выполняемых на одном физическом ресурсе.

Примером использования виртуализации является возможность запуска нескольких операционных систем на одном компьютере, притом каждый из экземпляров таких гостевых операционных систем работает со своим набором логических ресурсов (процессорных, оперативной памяти, устройств хранения), предоставлением которых из общего пула, доступного на уровне оборудования, управляет хостовая операционная система или гипервизор. Также могут быть подвергнуты виртуализации сети передачи данных, сети хранения данных, платформенное и прикладное программное обеспечение.

IBM признала важность виртуализации еще в 1960-х вместе с развитием компьютеров класса «мэйнфрэйм». Например, System/360 Model 67 виртуализировала все интерфейсы оборудования через программу Virtual Machine Monitor (VMM). VMM запускается непосредственно на основном оборудовании, позволяющем создавать множество виртуальных машин (VM), то есть стало возможным, например, запускать одну операционную систему на другой. При этом каждая виртуальная машина может обладать своей собственной операционной системой.

Когда производится виртуализация, существует несколько способов ее осуществления, с помощью которых достигаются одинаковые результаты через разные уровни абстракции. У каждого способа есть свои достоинства и недостатки, но главное что каждый из них находит свое место в зависимости от области применения. Можно считать, что самая сложная виртуализация обеспечивается эмуляцией аппаратных средств. В этом методе VM аппаратных средств создается на хост-системе, чтобы эмулировать интересующее оборудование.

Другое интересное использование эмуляции – это эмуляция оборудования, которая заключается в совместном развитии встроенного программного обеспечения и аппаратных средств. В этом методе VM аппаратных средств создается на хост-системе, чтобы эмулировать интересующее оборудование.

Эмуляция оборудования использует VM, чтобы моделировать необходимые аппаратные средства.

Вместо того чтобы дожидаться, когда реальные аппаратные средства будут в наличии, разработчики встроенного программного обеспечения могут использовать виртуальное оборудование для разработки и тестирования программного обеспечения.

Главная проблема при эмуляции аппаратных средств состоит существенном замедлении выполнения программ в такой среде. Поскольку каждая команда должна моделироваться на основных аппаратных средствах, при этом замедление в 100 раз при эмуляции является обычным делом. Однако эмуляция аппаратных средств имеет существенные преимущества. Например, используя эмуляцию аппаратных средств, можно управлять неизмененной операционной системой, предназначенной для PowerPC на системе с ARM процессором. также можно управлять многочисленными виртуальными машинами, каждая из которых будет моделировать другой процессор.

Полная (аппаратная) виртуализация, или «родная» виртуализация, является другим способом виртуализации. Эта модель использует менеджер виртуальных машин (гипервизор), который осуществляет связь между гостевой операционной системой и аппаратными средствами системы. Полная виртуализация, использует гипервизор, чтобы разделять основные аппаратные средства

Паравиртуализация — это другой популярный способ, который имеет некоторые сходства с полной виртуализацией. Этот метод использует гипервизор для разделения доступа к основным аппаратным средствам, но объединяет код, касающийся виртуализации, в непосредственно операционную систему. Этот подход устраняет потребность в любой перекомпиляции или перехватывании, потому что сами операционные системы кооперируются в процессе виртуализации. По сути, она разделяет процесс с гостевой ОС.

Виртуализация уровня операционной системы. Эта техника виртуализирует серверы непосредственно над операционной системой. Этот метод поддерживает единственную операционную систему и, в самом общем случае, просто изолирует независимые виртуальные серверы (контейнеры) друг от друга. Для разделения ресурсов одного сервера между контейнерами, данная виртуализация требует внесения изменений в ядро операционной системы (например, как в случае с OpenVZ), но при этом преимуществом является родная производительность, без «накладных расходов» на виртуализацию устройств. Виртуализация уровня ОС изолирует виртуальные сервиры.

77. Типы и реализации гипервизоров.

Гипервизор- программа или аппаратная схема, обеспечивающая или позволяющая одновременное, параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере. Гипервизор также обеспечивает изоляцию операционных систем друг от друга, защиту и безопасность, разделение ресурсов между различными запущенными ОС и управление ресурсами.

Типы гипервизора

Автомномный гипервизор (Тип 1)

Имеет свои встроенные драйверы устройств, модели драйверов и планировщик и поэтому не зависит от базовой ОС. Так как автономный гипервизор работает непосредственно в окружении усечённого ядра, то он более производителен, но проигрывает в производительности виртуализации на уровне ОС и паравиртуализации. Xen(автономный гипервизор) может запускать виртуальные машины в паравиртуальном режиме (зависит от ОС).

Примеры: VMware ESX, Citrix XenServer.

На основе базовой ОС (Тип 2, V)

Это компонент, работающий в одном кольце с ядром основной ОС (кольцо 0). Гостевой код может выполняться прямо на физическом процессоре, но доступ к устройствам ввода-вывода компьютера из гостевой ОС осуществляется через второй компонент, обычный процесс основной ОС — монитор уровня пользователя.

Примеры: Microsoft Virtual PC, VMware Workstation, QEMU, Parallels, VirtualBox.

Гибридный (Тип 1+)

Гибридный гипервизор состоит из двух частей: из тонкого гипервизора, контролирующего процессор и память, а также работающей под его управлением специальной сервисной ОС в кольце пониженного уровня. Через сервисную ОС гостевые ОС получают доступ к физическому оборудованию.

Примеры: Microsoft Virtual Server, Sun Logical Domains, Xen, Citrix XenServer, Microsoft Hyper-V.

Подходы к раелизации.

Монолитный.

Монолитный подход в реализации гипервизора подразумевает, что все драйвера устройств помещены в гипервизор. Вроде бы это дает некое преимущество с точки зрения безопасности драйверов. Нет возможности доработать драйверы, соответственно, никакой сторонний код не попадает в гипервизор.

В монолитной (monolithic) модели – гипервизор для доступа к оборудованию использует собственные драйверы. Гостевые ОС работают на виртуальных машинах поверх гипервизора. Когда гостевой системе нужен доступ к оборудованию, она должна пройти через гипервизор и его модель драйверов. Обычно одна из гостевых ОС играет роль администратора или консоли, в которой запускаются компоненты для предоставления ресурсов, управления и мониторинга всех гостевых ОС, работающих на сервере.

Модель монолитного гипервизора обеспечивает прекрасную производительность, но «хромает» с точки зрения защищенности и устойчивости. Это связано с тем, что она обладает более широким фронтом нападения и подвергает систему большему потенциальному риску, поскольку разрешает

работу драйверов (а иногда даже программ сторонних производителей) в очень чувствительной области.

Скажем, вредоносная программа способна жить в гипервизоре под видом драйвера устройства. Если такое случиться, под контролем такой программы окажутся все гостевые ОС системы, что, конечно, не радует. Хуже того, этот «жучок» будет совершенно невозможно обнаружить средствами гостевых ОС: гипервизор ими по определению не видим.

Микроядерный подход.

Альтернативу монолитному подходу в реализации гипервизора составляет микроядерная (microkernelized) реализация. В ней можно говорить о «тонком гипервизоре», в этом случае в нем совсем нет драйверов. Да-да, именно так – вообще нет драйверов. Вместо этого драйверы работают в каждом индивидуальном разделе, чтобы любая гостевая ОС имела возможность получить через гипервизор доступ к оборудованию. При такой расстановке сил каждая виртуальная машина занимает совершенно обособленный раздел, что положительно сказывается на защищенности и надежности.

В микроядерной модели гипервизора (в виртуализации Windows Server 2008 R2 используется именно она) один раздел является родительским (parent), остальные – дочерними (child). Раздел – это наименьшая изолированная единица, поддерживаемая гипервизором.

Каждому разделу вы назначаете конкретные аппаратные ресурсы – долю процессорного времени, объем памяти, устройства и пр. Родительский раздел создает дочерние разделы и управляет ими, а также содержит стек виртуализации (virtualization stack), используемый для управления дочерними разделами. Родительский раздел, вообще говоря, является также корневым (root), поскольку он создается первым и владеет всеми ресурсами, не принадлежащими гипервизору. Обладание всеми аппаратными ресурсами означает среди прочего, что именно корневой (то есть, родительский) раздел управляет питанием, подключением самонастраивающихся устройств, ведает вопросами аппаратных сбоев и даже управляет загрузкой гипервизора.

В родительском разделе содержится стек виртуализации – набор программных компонентов, расположенных поверх гипервизора и совместно с ним обеспечивающих работу виртуальных машин. Стек виртуализации обменивается данными с гипервизором и выполняет все функции по виртуализации, не поддерживаемые непосредственно гипервизором. Большая часть этих функций связана с созданием дочерних разделов и управлением ими и необходимыми им ресурсами (ЦП, память, устройства).

Стек виртуализации также обеспечивает доступ к интерфейсу управления, который в случае Windows Server 2008 R2 является поставщиком WMI.

Преимущество микроядерного подхода, примененного в Windows Server 2008 R2, по сравнению с монолитным подходом состоит в том, что драйверы, которые должны располагаться между родительским разделом и физическим сервером, не требуют внесения никаких изменений в модель драйверов. Иными словами, в системе можно просто применять существующие драйверы. В

Microsoft этот подход избрали, поскольку необходимость разработки новых драйверов сильно затормозила бы развитие системы. Что же касается гостевых ОС, они будут работать с эмуляторами или синтетическими устройствами.

78.Действия ОС при порождении процесса.

Новый процесс создается в UNIX только путем системного вызова **fork.** Процесс, сделавший вызов **fork**, называется *родительским*, а вновь созданный процесс - *порожденным*. Новый процесс является точной копией родительского. При порождении (разветвлении) процесса проверяется, достаточно ли памяти и места в таблице процессов для данного процесса. Если да, то образ текущего процесса копируется в новый образ процесса, и в таблице процессов возникает новый элемент. Новому процессу присваивается новый уникальный идентификатор (**PID**). Когда изменение таблицы процессов ядра завершается, процесс добавляется к списку процессов, доступных для выполнения и ожидающих в очереди планировщика подобно другим процессам.

Порожденный процесс **наследует** от родительского процесса следующие основные характеристики:

1. Способы обработки сигналов (адреса функций обработки сигналов).
2. Реальные и эффективные идентификаторы пользователя и группы.
3. Значение поправки приоритета.
4. Все присоединенные разделяемые сегменты памяти.
5. Идентификатор группы процессов.
6. Терминальную линию.
7. Текущий каталог.
8. Корневой каталог.
9. Маска создания файлов (**umask**).
10. Ограничения ресурсов (**ulimit**).

Порожденный процесс **отличается** от родительского процесса следующими основными характеристиками:

1. Порожденный процесс имеет свой уникальный идентификатор.
2. Порожденный процесс имеет другой идентификатор родительского процесса, равный идентификатору породившего процесса.
3. Порожденный процесс имеет свои собственные копии дескрипторов файлов (в частности, стандартных потоков), открытых родительским процессом. Каждый дескриптор файла порожденного процесса имеет первоначально такое же значение текущей позиции в файле, что и соответствующий родительский.
4. У порожденного процесса обнуляются счетчики времени, потраченного системой для его обслуживания.

Системный вызов **fork** завершается неудачей и новый процесс не порождается, если:

- Создать процесс запрещает системное ограничение на общее количество процессов.
- Создать процесс запрещает системное ограничение на количество процессов у одного пользователя.
- Общее количество системной памяти, предоставленной для физического ввода-вывода, временно оказалось недостаточным.

При успешном завершении порожденному процессу возвращается значение **0,** а родительскому процессу возвращается идентификатор порожденного процесса. В случае ошибки родительскому процессу возвращается **-1**, новый процесс не создается и переменной **errno** присваивается код ошибки.

Обычно после порождения порожденный процесс выполняет системный вызов **exec**, перекрывающий сегменты текста и данных процесса новыми сегментами текста и данных, взятыми из указанного выполняемого файла. При этом аппаратный контекст процесса инициализируется заново.

Выполняемый файл состоит из заголовка, сегмента команд и сегмента данных. Данные (глобальные переменные) состоят из инициализированной и неинициализированной частей.

Если системный вызов **exec** закончился успешно, то он не может вернуть управление, так как вызвавший процесс уже заменен новым процессом. Возврат из системного вызова **exec** свидетельствует об ошибке. В таком случае результат равен **-1**, а переменной **errno** присваивается код ошибки.

79.Основные типы планирования потоков.

На протяжении существования процесса выполнение его потоков может быть многократно прервано и продолжено. Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации. Работа по определению того, в какой момент необходимо прервать выполнение текущего активного потока и какому потоку предоставить возможность выполняться, называется планированием. Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков.

Планирование потоков, по существу, включает в себя решение двух задач: определение момента времени для смены текущего активного потока; выбор для выполнения потока из очереди готовых потоков.

В большинстве операционных систем универсального назначения планирование осуществляется динамически (on-line), то есть решения принимаются во время работы системы на основе анализа текущей ситуации. ОС работает в условиях неопределенности - потоки и процессы появляются в случайные моменты времени и также непредсказуемо завершаются.

Динамические планировщики могут гибко приспосабливаться к изменяющейся ситуации и не используют никаких предположений о мультипрограммной смеси. Другой тип планирования — статический — может быть использован в специализированных системах, в которых весь набор одновременно выполняемых задач определен заранее, например в системах реального времени. Планировщик называется статическим (или предварительным планировщиком), если он принимает решения о планировании не во время работы системы, а заранее (off-line).

80.Диспетчеризация потоков. Очереди потоков. Состав и организация контекста потока

Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой. Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем чтобы впоследствии использовать эту информацию для последующего возобновления выполнения данного потока. Диспетчеризация сводится к следующему: сохранение контекста текущего потока, который требуется сменить; загрузка контекста нового потока, выбранного в результате планирования; запуск нового потока на выполнение.

81.Вытесняющие и не вытесняющие алгоритмы планирования

С самых общих позиций все множество алгоритмов планирования можно разделить на два класса: вытесняющие и невытесняющие алгоритмы планирования. Невытесняющие (non-preemptive) алгоритмы основаны на том, что активному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению поток. 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 применении вытесняющих алгоритмов планирования ОС получает полный контроль над вычислительным процессом, а при применении невытесняющих алгоритмов решения принимаются децентрализованно: активный поток определяет момент смены потоков, а ОС выбирает новый поток для выполнения.

При невытесняющем мультипрограммировании механизм планирования распределен между операционной системой и прикладными программами. Такой

механизм создает проблемы как для пользователей, так и для разработчиков приложений. Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Поэтому разработчики приложений для операционной среды с невытесняющей многозадачностью вынуждены, возлагая на себя часть функций планировщика, создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Подобный метод работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста.

Существенным преимуществом невытесняющего планирования является более высокая скорость переключения с потока на поток. Примером эффективного использования невытесняющего планирования являются файл-серверы NetWare 3.x и 4.x, в которых в значительной степени благодаря такому планированию достигнута высокая скорость выполнения файловых операций.

Почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT/2000, OS/2, VAX/VMS), реализованы вытесняющие алгоритмы планирования потоков (процессов). В последнее время дошла очередь и до ОС класса настольных систем, например OS/2 Warp и Windows 95/98. 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.

82. Алгоритмы планирования, основанные на квантовании.

В основе многих вытесняющих алгоритмов планирования лежит концепция квантования. В соответствии с этой концепцией каждому потоку поочередно для выполнения предоставляется ограниченный непрерывный период процессорного времени – квант. Смена активного потока происходит, в следующих случаях: - поток завершился и покинул систему; - произошла ошибка; - поток перешел в состояние ожидания; - исчерпан квант процессорного времени, отведенный данному потоку. Поток, который исчерпал свой квант, переводится в состояние «готовность» и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый поток из очереди готовых. Таким образом, ни один поток не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени.

Многозадачные ОС теряют некоторое количество процессорного времени для выполнения вспомогательных работ во время переключения контекстов задач. Затраты на эти вспомогательные действия зависят от величины кванта времени – чем больше квант, тем меньше суммарные накладные расходы,

связанные с переключением потоков. В зависимости от используемого алгоритма планирования, кванты, выделяемые потокам, могут быть одинаковыми для всех потоков или различными. Кванты, выделяемые одному потоку, могут быть фиксированной величины, а могут и изменяться в разные периоды жизни потока. Например, первоначально каждому потоку назначается достаточно большой квант, а величина каждого следующего кванта уменьшается до некоторой заранее заданной величины. В таком случае короткие задачи успевают выполняться в течение первого кванта, а длительные вычисления будут проводиться в фоновом режиме. Если следующий квант, выделяемый потоку, больше предыдущего, то это позволяет уменьшить накладные расходы времени на переключение задач в том случае, когда выполняется одновременно сразу несколько длительных задач. Некоторые потоки не используют выделенный квант времени полностью, например, из-за необходимости выполнить ввод или вывод данных. Алгоритм планирования может предоставить таким потокам в качестве компенсации привилегии в виде внеочередного последующего обслуживания. Алгоритмы планирования, основанные на приоритетах. Другой важной концепцией, лежащей в основе многих вытесняющих алгоритмов планирования, является приоритетное обслуживание. Приоритетное обслуживание предполагает наличие у потоков некоторой изначально известной характеристики – приоритета, на основании которой определяется порядок их выполнения. Приоритет – это число, характеризующее степень привилегированности потока при использовании ресурсов ВС, в частности процессорного времени: чем выше приоритет, тем выше привилегии, тем меньше времени будет проводить поток в очередях. Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что приоритет потока тем выше, чем больше (в арифметическом смысле) число, обозначающее приоритет. В других системах, наоборот, чем меньше число, тем выше приоритет. Приоритет может назначаться директивно администратором системы, либо вычисляться самой ОС по определенным правилам. Приоритет может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими. В большинстве операционных систем, поддерживающих потоки, приоритет потока непосредственно связан с приоритетом процесса, в рамках которого выполняется данный поток.

Существует две разновидности приоритетного обслуживания: - обслуживание с относительными приоритетами; - обслуживание с абсолютными приоритетами. В системах с относительными и абсолютными приоритетами выбор потока на выполнение из очереди готовых осуществляется одинаково: выбирается поток, имеющий наивысший приоритет. По-разному определяется момент смены активного потока. В системах с относительными приоритетами активный поток выполняется до тех пор, пока он сам не покинет процессор при наступлении одного из следующих событий: процесс завершился, ошибка, переход в состояние «ожидание». Если ни одно из перечисленных событий не наступило, активный поток выполняется от начала до конца (квантование отсутствует). В системах с абсолютными приоритетами, в отличие от

предыдущего случая, выполнение активного потока принудительно прерывается операционной системой, если в очереди готовых потоков появился поток, приоритет которого выше приоритета активного потока. При этом прерванный поток переходит в состояние «готовность».

В системах на основе относительных приоритетов, минимизируются затраты на переключения процессора с одной работы на другую, но могут возникать ситуации, когда одна задача занимает процессор долгое время. Для систем разделения времени и реального времени относительные приоритеты не подходят, а вот в системах пакетной обработки используются широко. В системах с абсолютными приоритетами время ожидания потока с самым высоким приоритетом в очередях может сведено к минимуму. Это делает планирование на основе абсолютных приоритетов подходящим для систем управления объектами, в которых важна быстрая реакция на событие. Смешанные алгоритмы планирования. В ОС общего назначения используются, как правило, смешанные алгоритмы планирования, основанные как на квантовании, так и на приоритетах. Каждому потоку, в этом случае, выделяется квант времени, но его величина и порядок выбора потоков из очереди готовых определяется приоритетами потоков. Например, в Windows NT приоритеты потоков могут динамически меняться системой в определенном диапазоне относительно заданного значения.

83. Схема назначения приоритетов в ОС Windows NT.

Windows NT поддерживает 32 уровня приоритетов, разделенных на два класса - класс реального времени и класс переменных приоритетов. Нити реального времени, приоритеты которых находятся в диапазоне от 16 до 31, являются более приоритетными процессами и используются для выполнения задач, критичных ко времени.

В Windows NT определено 4 класса приоритетов процессов:

- IDLE_PRIORITY_CLASS - уровень 4
- NORMAL_PRIORITY_CLASS - уровень 9 при интерактивной работе процесса (forground) и уровень 7 при работе в фоновом режиме (background)
- HIGH_PRIORITY_CLASS - уровень 13
- REALTIME_PRIORITY_CLASS - уровень 24

Большинство приложений либо не определяет класс приоритета процесса при его создании, либо устанавливает его в значение NORMAL_PRIORITY_CLASS. Класс IDLE - самый низкоприоритетный - хорошо использовать для работ, некритичных к скорости их выполнения, например, при наблюдении за состоянием системы или же при резервном копировании на ленту. Высокий приоритет (HIGH_PRIORITY_CLASS) следует использовать только тогда, когда это абсолютно необходимо, так как нити такого процесса будут выполняться всегда перед нитями процесса с нормальным приоритетом. В Windows NT с приоритетом HIGH работает процесс Task Manager. Обычно он находится в состоянии ожидания, но при нажатии комбинации клавиш Ctrl+Esc нить Task

Manager пробуждается и немедленно вытесняет любые нити обычных приложений. Приоритеты реального времени системными процессами Windows NT (и тем более офисными приложениями) не используются. Этот класс приоритетов нужно использовать только для систем реального времени, например, сбора данных от промышленных установок, управления движущимися объектами и т.п.

84.Смешанный алгоритм планирования в ОС Windows NT.

Во многих операционных системах алгоритмы планирования построены с использованием как концепции квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора потока из очереди готовых определяется приоритетами потоков. Именно так реализовано планирование в системе Windows NT, в которой квантование сочетается с динамическими абсолютными приоритетами. На выполнение выбирается готовый поток с наивысшим приоритетом. Ему выделяется квант времени. Если во время выполнения в очереди готовых появляется поток с более высоким приоритетом, то он вытесняет выполняемый поток. Вытесненный поток возвращается в очередь готовых, причем он становится впереди всех остальных потоков имеющих такой же приоритет.

Смена активного потока происходит, если: • поток завершился и покинул систему; • произошла ошибка; • поток перешел в состояние ожидания; • исчерпан квант, отведенный данному потоку. Поток, исчерпавший свой квант, переводится в состояние готовности, активным делается новый поток из очереди готовых к выполнению. Типичное значение кванта в системах разделения времени (РДВ) – десятки миллисекунд. При этом потокам могут выделяться одинаковые или разные кванты, величина кванта данного потока может быть постоянной (фиксированной) или переменой по мере его развития. Переменная величина кванта может возрастать или убывать. Учет отмеченных факторов увеличивает число возможных вариантов алгоритмов планирования. При переменной величине кванта ее убывание выгодно коротким задачам, а возрастание позволяет уменьшить накладные расходы на переключение задач, если сразу несколько задач выполняют длительные вычисления. При интенсивном вводе-выводе поток не использует квант полностью. Такую дискриминацию можно компенсировать организацией дополнительной более приоритетной очереди готовых потоков, куда помещаются только потоки, прерванные из-за необходимости ввода- вывода. Многозадачные ОС теряют некоторое количество процессорного времени для выполнения вспомогательных работ во время переключения контекстов задач. При этом запоминаются и восстанавливаются регистры процессора, флаги и указатели стека, а также проверяется статус задач для передачи управления. Затраты на эти вспомогательные действия не зависят от величины кванта

времени, поэтому чем больше квант, тем меньше суммарные накладные расходы на переключение потоков. В алгоритмах планирования, основанных на квантовании, не используется никакой предварительной информации об особенностях решаемых задач (длительность, потребность в обмене, важность), подобная информация может накапливаться уже в ходе решения.

85.Алгоритм планирования в ОС UNIX System V Release 4

В алгоритме планирования в ОС UNIX System V Release 4.2 вообще отсутствуют потоки и поддерживается вытесняющая многозадачность, основанная приоритетах и квантовании. Здесь процесс может относиться к одному из классов: РВ (диапазон значений приоритетов 100-159) или РДВ (0-99), причем процессы РДВ делятся на пользовательские (0-59) и системные (60-99). Назначение и обработка приоритетов выполняются для процессов разных классов по- разному. Процессы системные РДВ, зарезервированные для ядра, используют стратегию фиксированных приоритетов. Их значения задаются ядром и никогда не изменяются. Процессы РВ также имеют фиксированные приоритеты, но пользователь может изменять их значения. Процессы РВ могут надолго захватывать процессор. Их характеризуют две величины: уровень глобального приоритета и квант. Для каждого уровня приоритета по умолчанию имеется своя величина кванта, что задается специальной внутренней таблицей ОС. По умолчанию новый процесс становится пользовательским РДВ. Состав процессов этого класса является наиболее неопределенным и часто меняющимся. Поэтому для справедливого распределения процессора 59 между процессами этого класса используется стратегия динамических приоритетов. Величина приоритета пользовательского процесса РДВ вычисляется пропорционально значениям двух частей: пользовательской и системной. Причем, результат суммирования принудительно ограничивается значением , чтобы процесс не становился системным РДВ. Пользовательская часть может быть изменена администратором или владельцем процесса (в последнем случае только снижена). Системная часть приоритета позволяет планировщику управлять процессами, имеющими долгие кванты. Приоритет таких процессов снижается, а процессов, часто не использующих свои кванты из-за ожидания окончания обмена, – повышается. Ущемленным (реже активизируемым) процессам с низкими приоритетами в компенсацию даются большие кванты, чем процессам с высокими приоритетами.

86.Смешанный алгоритм планирования в OS/2

В операционной системе OS/2 планирование основано на использовании квантования и абсолютных динамических приоритетов. Благодаря такому алгоритму планирования в OS/2 ни один поток не будет «забыт» системой и получит достаточно процессорного времени.

Все потоки разделены на 4 класса (по убыванию приоритета):

- критический (time critical). Например, системные потоки, решающие задачи управления сетью;
- серверный (server) – потоки, обслуживающие серверные приложения;
- стандартный (regular) – потоки обычных приложений;
- остаточный (idle) – наименее важные. Например, это поток, выводящий на экран заставку, когда в системе не выполняется никакой работы.

В каждом классе имеется 32 уровня (значения) приоритета, что дает в сумме 128 уровней. Поток из менее приоритетного класса не может быть активизирован, пока в очереди более приоритетного класса имеется хотя бы один поток. Внутри каждого класса потоки выбираются также по приоритетам. Потоки с равными приоритетами обслуживаются в циклическом порядке (по карусельной схеме).

Приоритеты потоков могут изменяться планировщиком OS/2 в следующих случаях:

- если поток находится в состоянии готовности дольше, чем это задано

системной переменной MAXWAIT, то OS/2 увеличит уровень его приоритета. При этом результирующее значение приоритета не должно переводить данный поток в критический класс;

- если поток начинает операцию ввода-вывода, то после ее завершения он получит наивысшее значение приоритета своего класса;
- при активизации потока его приоритет автоматически повышается.

87. Схема изменения приоритетов потоков и величины квантов при планировании в OS/2.

Планирование в ОС OS/2 также основано на использовании квантования и абсолютных динамических приоритетов [1-6, 11]. Все потоки разделены на 4 класса (по убыванию приоритета):

- критический (time critical). Например, системные потоки, решающие задачи управления сетью;
- серверный (server) – потоки, обслуживающие серверные приложения;
- стандартный (regular) – потоки обычных приложений;
- остаточный (idle) – наименее важные.

В каждом классе имеется 32 уровня (значения) приоритета, что дает в сумме 128 уровней. Поток из менее приоритетного класса не может быть активизирован, пока в очереди более приоритетного класса имеется хотя бы один поток. Внутри каждого класса потоки выбираются также по приоритетам. Потоки с равными приоритетами обслуживаются в циклическом порядке.

Приоритеты потоков могут изменяться планировщиком OS/2 в следующих случаях:

- если поток находится в состоянии готовности дольше, чем это задано 60 системной переменной MAXWAIT, то OS/2 увеличит уровень его приоритета. При этом результирующее значение приоритета не должно переводить данный поток в критический класс;
- если поток начинает операцию ввода-вывода, то после ее завершения он

получит наивысшее значение приоритета своего класса;

- при активизации потока его приоритет автоматически повышается. OS/2 динамически устанавливает и величину кванта потока в зависимости от степени загрузки системы и интенсивности подкачки. Средства настройки параметров OS/2 позволяют пользователю самостоятельно явно задать границы изменения величины кванта командой TIMESLICE=a, b (где a-b – задаваемый диапазон в пределах 32-65536 мс, по умолчанию 32-246 мс). Если поток был прерван до истечения кванта, то следующий квант его выполнения будет увеличен на 32 мс (один период таймера) и так много раз до тех пор, пока не будет достигнуто установленное значение b. Это позволяет уменьшить накладные расходы на переключение потоков, если в ВС несколько потоков одновременно ведут длительные вычисления. Благодаря такому алгоритму планирования в OS/2 ни один поток не будет «забыт» системой и получит достаточно процессорного времени.

88. События, требующие перераспределения процессорного времени, и действия планировщика ОС в каждом случае.

Когда дальнейшее выполнение потока невозможно, например из-за его завершения или перехода в ждущее состояние, ядро напрямую обращается к диспетчеру, чтобы вызвать немедленное переключение контекста. Однако иногда ядро обнаруживает, что перераспределение процессорного времени (rescheduling) должно произойти при выполнении глубоко вложенных уровней кода. В этой ситуации ядро запрашивает диспетчеризацию, но саму операцию откладывает до выполнения текущих действий. Такую задержку удобно организовать с помощью программного прерывания DPC (deferred procedure call).

При необходимости синхронизации доступа к разделяемым структурам ядра последнее всегда повышает IRQL процессора до уровня «DPC/dispatch» или выше. При этом дополнительные программные прерывания и диспетчеризация потоков запрещаются. Обнаружив необходимость в диспетчеризации, ядро генерирует прерывание уровня «DPC/dispatch». Но поскольку IRQL уже находится на этом уровне или выше, процессор откладывает обработку этого прерывания. Когда ядро завершает свои операции, оно определяет, что должно последовать снижение IRQL ниже уровня "DPC/dispatch", и проверяет, не ожидают ли выполнения отложенные прерывания диспетчеризации. Если да, IRQL понижается до уровня «DPC/dispatch», и эти отложенные прерывания обрабатываются. Активизация диспетчера потоков через программное прерывание — способ отложить диспетчеризацию до подходящего момента. Однако Windows использует программные прерывания для отложенного выполнения и других операций.

89. Моменты перепланировки в среде ОС РВ.

Для реализации своего алгоритма планирования ОС должна получать управление всякий раз, когда в системе происходит событие, требующее перераспределения процессорного времени:

1) прерывание от таймера – конец кванта. Планировщик ОС переводит задачу

(поток) в состояние готовности и выполняет перепланировку;
2) активная задача выполнила системный вызов – запрос на ввод-вывод или на доступ к ресурсу, который в настоящий момент времени занят. Планировщик переводит задачу в состояние ожидания и выполняет перепланировку;
3) активная задача 1 выполнила системный вызов, связанный с освобождением ресурса. Планировщик проверяет, не ожидает ли этот ресурс задача 2.
4) внешнее аппаратное прерывание, сигнализирующее о завершении УВВ операции ввода-вывода. Планировщик переводит соответствующую задачу в очередь готовых и выполняет перепланировку;
5) внутреннее прерывание, сигнализирующее об ошибке в результате выполнения активной задачи. Планировщик снимает задачу и выполняет перепланировку.

При возникновении каждого из этих событий планировщик ОС просматривает очереди задач и решает, какая задача будет выполняться следующей. Кроме отмеченных, существует и ряд других событий, часто связанных с системными вызовами, требующих перепланировки. Например, запросы приложений и пользователей на создание новой задачи или повышение приоритета уже существующей создают новую ситуацию, требующую пересмотра очередей и переключения процессора.

90. Диспетчеризация и учет приоритетов прерываний в ОС. Диспетчер прерываний.

Для упорядочения работы обработчиков прерываний в ОС применяется тот же механизм, что и для пользовательских процессов – механизм приоритетных очередей. Все источники прерываний обычно делятся на несколько классов, причем каждому классу присваивается приоритет. В ОС выделяется программный модуль, который занимается диспетчеризацией обработчиков прерываний – диспетчер прерываний, обеспечивающий обслуживание с абсолютными приоритетами. При возникновении прерывания диспетчер прерываний вызывается первым, запрещает ненадолго все прерывания, выясняет причину (и источник) прерывания. Затем диспетчер прерываний сравнивает назначенный данному источнику прерывания приоритет с текущим приоритетом потока команд, выполняемого процессором, и если новый приоритет выше текущего, запускает соответствующий обработчик. В этот момент времени процессор уже может выполнять инструкции другого обработчика прерываний, также имеющего некоторый приоритет. И если приоритет нового запроса выше текущего, то выполнение текущего обработчика приостанавливается, и он помещается в соответствующую очередь обработчиков прерываний. Иначе в очередь помещается обработчик нового запроса. Приоритет обработчиков прерываний в общем случае всегда выше приоритета потоков, выполняемых в обычной последовательности, определяемой планировщиком потоков. Поэтому любой запрос на прерывание всегда может прервать выполнение обычного потока. Так как процедуры, вызываемые по запросам прерываний, обычно выполняют работу, не связанную с текущим процессом, то для них вводятся ограничения: они не имеют права использовать ресурсы процесса или от его имени запрашивать выделение дополнительных ресурсов. Ресурсы обработчиков прерываний принадлежат ОС, а не конкретному процессу. Поэтому обычно говорят, что

процедуры обработки прерываний работают вне контекста процесса, хотя бывают и исключения (Windows NT). Диспетчеризация прерываний является важной функцией, 64 реализованной практически во всех мультипрограммных ОС. В общем случае в ОС реализуется двухуровневый механизм планирования работ. Верхний уровень планирования выполняется диспетчером прерываний, который распределяет процессор между потоком поступающих запросов на прерывания различных типов – внешних, внутренних и программных. Оставшееся процессорное время распределяется другим диспетчером – диспетчером потоков, на основании дисциплин квантования и других.

91. Диспетчеризация системных вызовов.

Системный вызов позволяет приложению обратиться к ОС с просьбой выполнить то или иное действие, оформленное как процедура (или набор процедур) кодового сегмента ОС. Реализация системных вызовов должна:

- обеспечивать переключение в привилегированный режим (часто только с помощью механизма программных прерываний);
- обладать высокой скоростью вызова процедур ОС;
- обеспечивать по возможности единообразное обращение к системным вызовам;
- допускать легкое расширение набора системных вызовов;
- обеспечивать контроль со стороны ОС за корректным использованием системных вызовов.

Здесь при любом системном вызове приложение выполняет программное прерывание (INT) с определенным и единственным номером вектора. Перед выполнением программного прерывания приложение некоторым образом передает ОС номер системного вызова, который является индексом в специально организованной дополнительной таблице sysent адресов процедур ОС, реализующих системные вызовы. Кроме номера передаются аргументы системного вызова. После завершения обработки системного вызова управление возвращается диспетчеру системных вызовов вместе с кодом завершения этого вызова. Диспетчер системных вызовов восстанавливает регистры процессора, помещает в определенный регистр код возврата и выполняет инструкцию возврата из прерывания, которая восстанавливает непривилегированный режим работы процессора.

92. Схема организации системных вызовов с диспетчером системных вызовов.

Диспетчер системных вызовов обычно представляет собой простую программу, которая сохраняет содержимое регистров процессора в системном стеке (в связи со сменой режима процессора), проверяет, попадает ли запрошенный номер вызова в поддерживаемый ОС диапазон (в таблицу sysent адресов системных вызовов) и передает управление адресованной процедуре ОС. Искомая процедура обработки системного вызова извлекает из системного стека аргументы и выполняет заданное действие. После завершения обработки системного вызова управление возвращается диспетчеру системных вызовов вместе с кодом завершения этого вызова. Диспетчер системных вызовов

восстанавливает регистры процессора, помещает в определенный регистр код возврата и выполняет инструкцию возврата из прерывания, которая восстанавливает непривилегированный режим работы процессора.

93. Особенности и различия организации синхронных и асинхронных системных вызовов.

ОС может выполнять системные вызовы в синхронном или асинхронном режиме. Поток (процесс), сделавший синхронный (блокирующий) системный вызов, переводится планировщиком ОС в состояние ожидания, а после завершения обработки вызова – в состояние готовности. Тогда при следующей активизации этот поток уже сможет воспользоваться результатами своего системного вызова. В свою очередь, асинхронный системный вызов не приводит к переходу потока в состояние ожидания результатов вызова. Вместо этого поток переводится в состояние готовности, пока активизируются короткие начальные системные действия (например, запуск операции ввода-вывода), но неясно, когда поток сможет воспользоваться результатами данного системного вызова (это становится заботой самого потока). Большинство системных вызовов в ОС являются синхронными, так как это избавляет приложение от работы по выяснению момента появления результатов вызова. Вместе с тем в новых версиях ОС число асинхронных системных вызовов постепенно растет, что дает больше свободы разработчикам сложных приложений. Особенно нужны асинхронные системные вызовы в ОС на основе микроядра, так как при этом в пользовательском режиме работает часть модулей ОС, которым необходимо иметь полную свободу в организации своей работы, а такую свободу дает только асинхронный режим обслуживания вызовов микроядром.

94. Цели взаимодействия, синхронизация процессов и потоков. Гонки, критическая секция, взаимное исключение потоков.

В мультипрограммной ОС процессы (потоки) также могут и взаимодействовать друг с другом, причем с такими двумя целями как обмен данными и взаимная синхронизация исполнения. Организация осложняется тем, что оно происходит в условиях разделения (совместного использования)

взаимодействующими процессами и потоками аппаратных и информационных ресурсов ВС. Синхронизация необходима для исключения дефектов (гонок и тупиков) при обмене данными между потоками, разделении данных, при доступе к процессору и к УВВ. Любое взаимодействие процессов или потоков связано с их синхронизацией, которая заключается в согласовании их скоростей путем приостановки отдельного потока до наступления некоторого события с последующей активизацией данного потока при наступлении этого события. Синхронизация требуется независимо от того, с чем связано взаимодействие (с разделением ресурсов или с обменом данными), так как процессы или потоки связаны отношением «производитель-потребитель».

Ситуации, когда два или более потоков обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей потоков, называются гонками.

Критическая секция – это часть программы, результат выполнения которой может непредсказуемо меняться, если переменные, относящиеся к ней, изменяются другими потоками в то время, когда ее выполнение еще не завершено. Критическая секция всегда определяется по отношению к определенным критическим данным, при несогласованном изменении которых могут возникнуть нежелательные эффекты.

Для исключения эффекта гонок по отношению к критическим данным необходимо обеспечить взаимное исключение, чтобы в каждый момент времени в критической секции находился только один поток, неважно в каком состоянии. При этом обычно ОС использует разные способы реализации взаимного исключения, пригодные для потоков одного и/или разных процессов.

95. Блокирующие переменные и семафоры. Примеры.

Блокирующие переменные. Для синхронизации потоков одного процесса прикладной программист может использовать глобальные блокирующие переменные, доступные всем его потокам. Каждому набору критических данных D ставится в соответствие двоичная переменная F, которой поток присваивает значение F(D)=0 при вхождении в критическую секцию или F(D)=1 – при выходе из нее. Пока F(D)=0, другой поток, циклически проверяя это, не может войти в

критическую секцию

Семафоры. Обобщением (двоичных) блокирующих переменных являются семафоры Д. Дийкстры – переменные, принимающие целые неотрицательные значения.

Для работы с семафорами (S) вводятся два примитива, традиционно обозначаемых P и V, и операции

- **V(S)**: S:=S+1 (здесь ':=' – знак «присвоить»);
- **P(S)**: если S>0, то S:=S–1, иначе поток, вызывающий операцию P(S), ожидает момента, когда станет S>0.

Никакие прерывания и доступ к S во время выполнения примитивов V(S) и P(S) недопустимы. Частным случаем рассмотренного является двоичный семафор (блокирующая переменная). Операция P(S) способствует переходу вызывающего его потока в состояние ожидания, операция V(S) – активизации приостановленного операцией P(S) потока.

96. Пример использования семафоров при работе с буферным пулом записи чтения.

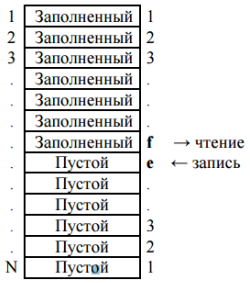


Рис.4.5. Работа с буферным пулом (чтение и запись)

Пусть пул включает N буферов, каждый буфер может содержать одну запись. В общем случае поток-писатель и поток-читатель могут обращаться к пулу с переменной интенсивностью. При этом скорость заполнения пула может превышать скорость освобождения и наоборот. Тогда для правильной совместной работы поток-писатель должен приостанавливаться при заполнении пула (в состоянии, когда некуда писать), а поток-читатель – при освобождении пула (в состоянии, когда нечего читать). Введем два семафора: e – число пустых (empty) буферов, f – число заполненных (filled) буферов. N=e + f. В исходном состоянии e = N, f = 0. Поток-писатель выполняет операцию P(e): если e>0, то записывает данные в пустой буфер, делает уменьшение e:=e–1 и выполняет операцию V(f): f:=f+1; иначе ожидание. Поток-читатель выполняет операцию P(f): если f>0, то читает данные из непустого буфера, делает уменьшение f:=f–1 и выполняет операцию V(e): e:=e+1; иначе ожидание. Заметим, что буферный пул здесь является критическим ресурсом. А использование блокирующей переменной вместо семафора не позволяет 72 организовать доступ к критическому ресурсу более чем одному потоку. Последнее актуально, если требуется сделать запись и чтение данных критическими секциями и обеспечить взаимное исключение. Семафор же решает задачу более гибко, допуская к разделяемому пулу ресурсов

заданное число потоков. В нашем примере с пулом могут работать до N потоков, часть из которых – «писатели», остальные – «читатели».

97. Тупики, идеи и средства их выявления и устранения.

Взаимные блокировки (дедлоки, клинчи или **тупики)** – ситуации, когда два и более потоков из-за занятости, запретов или ограничений доступа к ресурсам (памяти, УВВ) могут взаимно и неразрешимо мешать развитию друг друга.

На случаи, когда тупики все же возникают, администратору нужны средства их своевременного выявления, снятия и восстановления нормальной работы ВС. Тупики должны предотвращаться при написании приложений. Другой подход заключается в том, что ОС при каждом запуске задач анализирует состав мультипрограммной смеси (требования задач к ресурсам). Запуск задачи, которая может вызывать тупик, временно откладывается. ОС может также использовать особые правила выделения ресурсов процессам, например, строго последовательно и полностью.

Тупиковую ситуацию важно быстро и точно распознать. Если тупик образован множеством потоков, занимающих массу ресурсов, распознавание его становится 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 обнаруживать взаимные блокировки алгоритмически.

Если же тупиковая ситуация все же возникла, не обязательно снимать с решения все задачи или заблокированные потоки. Достаточно начать снимать некоторые из них, освобождая ожидаемые ресурсы. Можно вернуть некоторые потоки в область подкачки. Можно совершить откат некоторых потоков до так называемой контрольной точки, специально созданной программистом, где запоминается вся информация для восстановления выполнения программы с данного места и т.п.

98. Виды синхронизирующих объектов ОС, их сигнальное состояние, примеры.

Конкретный смысл сигнального состояния зависит от типа объекта. Например, поток переходит в сигнальное состояние, когда он завершается, процесс – когда завершаются все его потоки, файл – когда завершается операция его ввода-вывода, остальные объекты – в результате выполнения специальных системных вызовов. Тогда приостановка и активизация потоков осуществляется в зависимости от состояния синхронизирующих объектов ОС.

Поток, выполнивший системный вызов Wait(X), переводится операционной системой в состояние ожидания до тех пор, пока объект X не перейдет в сигнальное состояние.

Примерами системных вызовов типа Wait() и Set() являются вызовы WaitForSingleObject() и SetEvent() в Windows NT, DosSemWait() и DosSemSet() в OS/2, sleep() и wakeup() в UNIX.

Рассмотрим еще два примера, где синхронизирующими объектами выступают процессы и файлы.

Пусть выполнение некоторого приложения требует последовательных работ (этапов). Для каждого этапа организован 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 процесс.

Объект-файл, переход которого в сигнальное состояние соответствует завершению операции ввода-вывода с этим файлом, используется в тех случаях, когда инициировавший эту операцию поток решает дождаться ее завершения, прежде чем продолжить свои вычисления.

99.Мьютекс, объект-событие. Сигналы.

1. **Мьютекс** (от английского mutually exclusive access – взаимно исключающий доступ, когда одновременный доступ к общему ресурсу исключается) **или взаимоисключение**, как и семафор, обычно используется для управления доступом к данным.
2. **Объект-событие** обычно используется для того, чтобы оповестить другие потоки о том, что некоторые действия завершены.
3. **Сигнал** дает возможность задаче реагировать на событие, источником которого может быть ОС или другая задача. Сигналы вызывают прерывание задачи и выполнение заранее предусмотренных действий. Сигналы могут вырабатываться синхронно (в результате работы самого процесса) или асинхронно (направляться процессу от другого процесса). Синхронные сигналы чаще всего приходят от системы прерываний процессора и свидетельствуют о действиях процесса, блокируемых аппаратурой, например, деление на 0, ошибка адресации, нарушение защиты памяти.

В системе может быть определен целый 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 коде необходимо предусмотреть специальные системные вызовы, с помощью которых ОС информируется о том, какую именно процедуру надо выполнить в ответ на поступление того или иного сигнала. Сигналы

обеспечивают логическую связь между процессами, а также между процессами и пользователями (терминалами). А 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 ОП, ранее рассмотренные средства (блокирующие переменные, семафоры, сигналы) и другие аналогичные средства, основанные на разделяемой памяти, к сожалению, оказываются непригодными. В таких системах синхронизация может быть реализована только посредством явного обмена сообщениями.

100.Виды адресов команд и данных. ВАП и образ процесса.

1) виртуальные (математические, логические) условные адреса, вырабатываемые транслятором, переводящим программу и все ее авторские символьные имена на машинный язык. Во время трансляции в общем случае неизвестно, в какое место ОП будет загружена программа. Поэтому транслятор может присвоить переменным и командам только виртуальные (условные) адреса, по умолчанию начиная программу с нулевого адреса;
2) физические адреса, соответствующие реальным номерам ячеек ОП, где могут быть расположены переменные и команды. Совокупность виртуальных адресов процесса называется его виртуальным адресным пространством (ВАП).

Содержимое назначенного процессу ВАП (коды команд, исходные и промежуточные данные, результаты вычислений) представляет собой **образ процесса**.
Во время работы процесса постоянно выполняются переходы от прикладных кодов к кодам ОС (например, в случае явного вызова системных функций, вызова как реакции на внешние события или исключительные ситуации в работе приложения и др.). Для упрощения передачи управления от прикладного кода к коду ОС, а также для легкого доступа модулей ОС к прикладным данным в большинстве ОС для каждого процесса сегменты ОС и его прикладные сегменты образуют единое ВАП. При этом обычно ВАП процесса делится на две непрерывные части: системную и пользовательскую. Их размер может быть разным, например, в Windows NT он составляет по 2 Гбайта для ОС и для приложений. Системная часть ВАП каждого процесса, отводимая под сегменты ОС, является идентичной для всех процессов. Поэтому при смене активного процесса заменяется только вторая пользовательская часть ВАП, содержащая его индивидуальные сегменты (коды и данные приложения).

101.Способы структурирования ВАП процесса.

В разных ОС используются различные способы структурирования ВАП:

1) плоская (flat) структура – в виде непрерывной линейной последовательности виртуальных адресов. Здесь адрес определяется как число m, задающее смещение относительно начала ВАП.

2) ВАП делится на части одного вида – сегменты (области и т.п.). Виртуальный адрес в этом случае представляет собой пару чисел вида (номер сегмента, смещение внутри сегмента).

3) ВАП делится на части нескольких видов, что усложняет адрес до нескольких чисел [1-2,11].

102.Классы алгоритмов распределения ОП, состав классов.

Все алгоритмы распределения ОП делятся на два класса, работающие
(1) без использования внешней памяти:

1)распределения фиксированными разделами. ОП на сеанс работы разбивается на несколько разделов фиксированной величины каждый. К разделам организуется одна или несколько очередей процессов,

2)распределения динамическими разделами, по мере порождения процессов. Алгоритм связан с различными вариантами поиска свободной области ОП для раздела, отличается большей гибкостью, но подвержен эффекту фрагментации,

3)распределения перемещаемыми разделами. Используется как метод борьбы с фрагментацией на основе ликвидации «дыр» в ОП;

(2) с использованием внешней памяти:

1)страничного распределения, когда частями ОП и ВАП являются страницы фиксированного и сравнительно небольшого размера,

2)сегментного распределения, когда частями ОП и ВАП являются сегменты произвольного размера (какой получится), полученные с учетом смыслового значения данных,

3)сегментно-страничного распределения – комбинация двух предыдущих алгоритмов, когда ВАП делится на сегменты, которые затем делятся на страницы [2].

103.Свопинг, его достоинства и недостатки.

1)**свопинг** (swapping – подкачка) – образы процессов выгружаются на диск и возвращаются в ОП целиком. Подкачке свойственна избыточность, так как часто достаточно было бы загрузить/выгрузить лишь часть образа процесса. Кроме того, перемещение избыточной информации замедляет работу системы, приводит к неэффективному использованию ОП. Нельзя загрузить для выполнения процесс, ВАП которого превышает размер свободной ОП
2) **виртуальная память** – между ОП и диском перемещаются части образов процессов (страницы, сегменты).

Для временного хранения сегментов и страниц на диске отводится либо специальная область, либо специальный файл, который по традиции продолжают называть областью или файлом свопинга (swap-файлом), хотя перемещение между ОП и диском осуществляется уже не процессами целиком, а их частями. Используется и другое название – страничный файл (page file, paging file). Текущий размер страничного файла является важным параметром: чем он больше, тем больше приложений может одновременно выполнять ОС при фиксированном размере ОП. Но при этом их работа замедляется, так как значительная часть времени тратится на перекачку кодов и данных из ОП на диск и обратно.
Из-за своих недостатков свопинг как основной механизм управления памятью почти не используется в современных ОС. Например, в версиях UNIX, основанных на System V Release 4, свопинг служит дополнением виртуальной памяти и включается лишь при серьезных перегрузках системы.

104.Страничное распределение ОП.

Размер ВАП процесса в общем случае не кратен размеру страницы. Размер страницы специально выбирается равным 2k байт, например, 2 9 = 512 байт, 2 10 = 1024 байт, 2 12 = 4096 байт = 4 Кбайта, что упрощает механизм преобразования адресов. При порождении процесса ОС загружает в ОП несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Страницы могут быть расположены в ОП не подряд. Копия всего ВАП находится на диске. Для каждого порождаемого процесса ОС создает в ОП таблицу страниц, содержащую записи о каждой виртуальной странице процесса – дескрипторы страниц.

Дескриптор страницы включает:

- номер физической страницы, куда загружена данная виртуальная страница;
- признак присутствия, равный 1, если данная виртуальная страница загружена в ОП;
- признак модификации, равный 1, если данная виртуальная страница была изменена и при выгрузке ее (как обновленную) надо будет скопировать на диск;
- признак обращения, равный 1 при каждом обращении к данной виртуальной странице.

С каждой страницей связан счетчик числа обращений. ОС периодически просматривает признаки обращения и обнуляет ненулевые значения, одновременно наращивая значение соответствующего счетчика. Чтобы учесть интенсивность обращений за последний период, ОС с соответствующей периодичностью обнуляет все счетчики. Таблица страниц нужна для решения вопроса о перемещении страниц, а также преобразования виртуального адреса в физический. Адрес самой таблицы страниц включается в контекст процесса, а при активизации процесса загружается в специальный регистр процессора [2]. При каждом обращении к памяти (по какому-то адресу) выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру

определяется нужный дескриптор страницы, и из него извлекается описывающая страницу информация. Если признак присутствия равен 1 (страница находится в ОП), то выполняется преобразование виртуального адреса в физический. В противном случае происходит страничное прерывание. Далее активный процесс переводится в состояние ожидания, активизируется один из других готовых процессов. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу в страничном файле и пытается загрузить ее в ОП. Если в ОП имеется свободная физическая страница, то загрузка выполняется немедленно. В противном случае на основе принятой в данной ОС специальной стратегии замещения страниц выбирается некоторая «ненужная» страница. Обнуляется ее признак присутствия и анализируется признак модификации. Если признак 90 модификации равен 1 (страница изменялась), вытесняемую (обновленную) страницу необходимо скопировать на ее место в образе процесса на диске, иначе искомая физическая страница просто объявляется свободной (в некоторых ОС для повышения надежности она может еще и обнуляться).

105.Известные стратегии замещения страниц.

Обычно рассматривают три стратегии:

1) *Стратегия выборки* (*fetch policy*) - в какой момент следует переписать страницу из вторичной памяти в первичную. Выборка бывает по запросу и с упреждением. Алгоритм выборки вступает в действие в тот момент, когда процесс обращается к не присутствующей странице, содержимое которой в данный момент находится на диске (в своп файле или отображенном файле), и потому является ключевым алгоритмом свопинга. Он обычно заключается в загрузке страницы с диска в свободную физическую страницу и отображении этой физической страницы в то место, куда было произведено обращение, вызвавшее исключительную ситуацию. Существует модификация алгоритма выборки, которая применяет еще и опережающее чтение (с упреждением), т.е. кроме страницы, вызвавшей исключительную ситуацию, в память также загружается несколько страниц, окружающих ее (так называемый кластер). Такой алгоритм призван уменьшить накладные расходы, связанные с большим количеством исключительных ситуаций, возникающих при работе с большими объемами данных или кода, кроме того, оптимизируется и работа с диском, поскольку появляется возможность загрузки нескольких страниц за одно обращение к диску.

2) *Стратегия размещения* (*placement policy*) - определить в какое место первичной памяти поместить поступающую страницу. В системах со страничной организацией в любой свободный страничный кадр (в системах с сегментной организацией - нужна стратегия, аналогичная стратегии с переменными разделами).

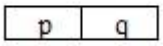
3) *Стратегия замещения* (*replacement policy*) - какую страницу нужно вытолкнуть во внешнюю память, чтобы освободить место. Разумная стратегия замещения

позволяет оптимизировать хранение в памяти самой необходимой информации и тем самым снизить частоту страничных нарушений.

106.Поддержка разделов при страничном распределении ОП.

Основные положения схемы страничного распределения:

- память разбивается на блоки фиксированной длины;
- адресное пространство любой задачи разбивается на страницы такой же длины;
- страницы одной задачи должны быть все одновременно в ОП, но могут занимать различные участки памяти;
- страницы не перемещаются;
- для каждой задачи организуется таблица страниц;
- структура адреса (исполнительного):

  где p - № страницы, q - смещение (от начала страницы). Число разрядов для q - определяет максимальный размер страницы 2q (размер страницы кратен 2). Обычно размер страницы 1К, 2К, 4К;
- используется "регистр адреса таблицы страниц" - для указания адреса таблицы страниц текущей задачи.

Таблица страниц устанавливает взаимосвязь между логическим страницами, с которыми работает процесс пользователя, и физическими страницами, с которыми работает центральный процессор.

С одной физической страницей могут быть связаны несколько логических (например: когда несколько процессов разделяют некоторую повторно входимую программу).

Логическое адресное пространство страницы - это источник информации для передачи данных (чему-либо - в наше случае - физической памяти).

Достоинства распределения:

- отсутствует внешняя фрагментация (а есть, например, внутри страницы);
- нет затрат на сжатие.

Недостатки распределения:

- двойное обращение к ОП - быстродействие понижается в, меры исключения этого недостатка: ассоциативная память, специальные регистры для всех таблиц страниц.
- накладные расходы на таблицы;
- внутренняя фрагментация ("усеченные страницы");
- задачи, занимающие ОП, не все используются;
- возможны отказы.

107.Сегментное распределение ОП.

При сегментной организации памяти ВАП процесса делится на части – сегменты, размер которых определяется с учетом смыслового значения содержащейся в них информации (подпрограмма, массив данных и т.п.). Деление ВАП на сегменты производится компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями. Размер сегмента ограничивается лишь разрядностью виртуального адреса, например, при 32-разрядной адресации сегмент может занимать до 4 Гбайт. В каждом сегменте виртуальные адреса находятся в диапазоне от 0000000016 до FFFFFFFF16. А ВАП процесса представляет собой набор N виртуальных сегментов, не упорядоченных друг относительно друга, как показано на рис.5.9, где базовые (начальные) физические адреса сегментов обозначены S1, S2, S3. При загрузке процесса в ОП там помещается только часть его 95 сегментов, а полная копия ВАП хранится в дисковой памяти. Для каждого загружаемого сегмента ОС подыскивает непрерывный участок ОП достаточного размера, что, конечно, вызывает большие проблемы по сравнению с поиском свободной страницы [1, 2]. Смежные сегменты ВАП процесса могут в ОП занимать несмежные участки. Если во время выполнения процесса происходит обращение по виртуальному адресу, относящемуся к сегменту, который в настоящий момент отсутствует в ОП, то происходит прерывание.
ОС переводит активный процесс в состояние ожидания, активизирует очередной процесс, а параллельно организует загрузку нужного сегмента с диска. А при отсутствии в ОП свободного места для загрузки данного сегмента, ОС выбирает «ненужный» сегмент для выгрузки по критериям, аналогичным критериям выгрузки страниц. При порождении процесса во время загрузки его образа в ОП ОС создает таблицу сегментов процесса, подобную таблице страниц и содержащую в дескрипторе каждого сегмента: базовый физический адрес сегмента в ОП; размер сегмента; правила доступа к сегменту; признаки: модификации, присутствия и обращения к сегменту, а также некоторую дополнительную информацию. Если ВАП нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок ОП, в который данный сегмент загружается в единственном FF…FF 00…00 FF…FF 00…00 FF…FF 00…00 S3 S2 S1 Оперативная память ВАП процесса ВАП Сегмента 1 ВАП Сегмента 2 ВАП Сегмента 3 Сегмент 2 Сегмент 1 Сегмент 397 экземпляре.

108.Сегментно-страничное распределение ОП.

Это комбинация страничного и сегментного механизмов управления памятью с реализацией достоинств обоих подходов:
• ВАП процесса разделено на сегменты, что обеспечивает поддержку разных прав доступа;
• перемещение данных между ОП и диском осуществляется страницами. Для этого каждый виртуальный сегмент и вся физическая память делятся на страницы равного (фиксированного) размера, что позволяет более эффективно использовать ОП, до минимума сокращая фрагментацию. Но в отличие от ВАП в виде набора отдельных виртуальных диапазонов адресов (0000000016 – FFFFFFFF16) каждого

сегмента при сегментной организации (рис.5.9), при сегментно-страничной организации все виртуальные сегменты образуют одно общее непрерывное линейное ВАП процесса в диапазоне (0000000016 – FFFFFFFF16), показанное на рис.5.11. Координаты байта в ВАП теоретически можно задать двумя способами:

- линейным виртуальным адресом, который равен сдвигу данного байта относительно границы общего линейного ВАП;
- парой чисел вида (номер сегмента, смещение от начала сегмента), а также указать начальный виртуальный адрес сегмента с данным номером.

Системы виртуальной памяти ОС с сегментно-страничной 99 организацией используют именно второй способ, как позволяющий непосредственно определить принадлежность адреса некоторому сегменту и проверить права доступа процесса к нему.

Первый этап. Работает механизм сегментации. Исходный виртуальный адрес вида (номер сегмента, смещение) преобразуется в линейный виртуальный адрес. Для этого по базовому адресу таблицы сегментов и номеру сегмента из таблицы сегментов извлекается дескриптор этого сегмента. Производится анализ полей дескриптора. Если доступ к сегменту разрешен, то вычисляется линейный виртуальный адрес путем сложения базового виртуального адреса сегмента (из дескриптора) и смещения.

Второй этап. Работает страничный механизм. Полученный линейный виртуальный адрес преобразуется в искомый физический адрес. Вначале выделяются его составляющие (номер виртуальной страницы, смещение в странице), понятные страничному механизму. Благодаря тому, что размер страницы равен 2k, эта задача решается простым отделением младших k двоичных разрядов. При этом смещением является содержимое младших k разрядов, а номером виртуальной страницы – содержимое оставшихся старших разрядов. Далее преобразование адреса в физический происходит так же, как при страничной организации. Номер виртуальной страницы дает адрес дескриптора страницы в таблице страниц, по которому из дескриптора выбирается номер физической страницы. Последний операцией конкатенации присоединяется к смещению.

109. Разделяемые сегменты памяти

Подсистема виртуальной памяти может обеспечивать и совместный доступ нескольких процессов к одному и тому же сегменту памяти, который в этом случае называется разделяемой памятью (shared memory). Разделяемый сегмент необходимо поместить в ВАП каждого
работающего с ним процесса, после чего настроить параметры отображения этих виртуальных сегментов так, чтобы они соответствовали одной и той же области ОП. Детали такой настройки зависят от используемой в ОС модели виртуальной памяти, поддерживающей сегменты (сегментной или сегментно-страничной).

Разделяемые сегменты памяти как средство межпроцессной связи позволяют процессам иметь общие области виртуальной памяти и, как следствие, разделять содержащуюся в них информацию. Единицей разделяемой памяти являются сегменты, свойства которых зависят от аппаратных особенностей управления памятью.

Разделение памяти обеспечивает наиболее быстрый обмен данными между процессами.

Работа с разделяемой памятью начинается с того, что процесс при помощи системного вызова shmget создает разделяемый сегмент, специфицируя первоначальные права доступа к сегменту (чтение и / или запись) и его размер в байтах. Чтобы затем получить доступ к разделяемому сегменту, его нужно присоединить посредством системного вызова shmat(), который разместит сегмент в виртуальном пространстве процесса. После присоединения, в соответствии с правами доступа, процессы могут читать данные из сегмента и записывать их (быть может, синхронизируя свои действия с помощью семафоров).

Когда разделяемый сегмент становится ненужным, его следует отсоединить, воспользовавшись системным вызовом shmdt().

Для выполнения управляющих действий над разделяемыми сегментами памяти служит системный вызов shmctl(). В число управляющих действий входит предписание удерживать сегмент в оперативной памяти и обратное предписание о снятии удержания. После того, как последний процесс отсоединил разделяемый сегмент, следует выполнить управляющее действие по удалению сегмента из системы.

110. Задачи ОС по управлению УВВ и файлами.

Одной из главных задач ОС является обеспечение обмена данными между работающими программами и УВВ компьютера. В современных ОС эти функции выполняет подсистема ввода-вывода, клиентами которой являются не только пользователи и приложения, но и компоненты самой ОС при необходимости получения или вывода ими каких-то системных данных.

Подсистема ввода-вывода (Input-Output Subsystem) мультипрограммной ОС при обмене данными с внешними устройствами компьютера должна решать ряд общих задач, из которых наиболее важными являются следующие:

- организация параллельной работы устройств ввода-вывода и процессора;
- согласование скоростей обмена и кэширование данных;

- разделение устройств и данных между процессами;
- обеспечение удобного логического интерфейса между устройствами и остальной частью системы;
- поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера;
- динамическая загрузка и выгрузка драйверов;
- поддержка нескольких файловых систем;
- поддержка синхронных и асинхронных операций ввода-вывода.

Рассмотрим перечисленные задачи более подробно.

Организация параллельной работы устройств ввода-вывода и процессора. Каждое устройство ввода-вывода вычислительной системы - диск, принтер, терминал и т. п. - снабжено специализированным блоком управления, называемым контроллером. Контроллер взаимодействует с драйвером - системным программным модулем, предназначенным для управления данным устройством. Контроллер периодически принимает от драйвера выводимую на устройство информацию, а также команды управления, которые говорят о том, что с этой информацией нужно сделать (например, вывести в виде текста в определенную область терминала или записать в определенный сектор диска). Под управлением контроллера устройство может некоторое время выполнять свои операции автономно, не требуя внимания со стороны центрального процессора.

Процессы, происходящие в контроллерах, протекают в периоды между выдачами команд независимо от ОС. От подсистемы ввода-вывода требуется спланировать в реальном масштабе времени (в котором работают внешние устройства) запуск и приостановку большого количества разнообразных драйверов, обеспечив приемлемое время реакции каждого драйвера на независимые события контроллера. При этом необходимо минимизировать загрузку процессора задачами ввода-вывода, оставив как можно больше процессорного времени на выполнение пользовательских потоков.

Согласование скоростей обмена и кэширование данных. При обмене данными всегда возникает задача согласования скорости. Например, если один пользовательский процесс вырабатывает некоторые данные и передает их другому пользовательскому процессу через оперативную память, то в общем случае скорости генерации данных и их чтения не совпадают. Согласование скорости обычно достигается за счет буферизации данных в оперативной памяти и синхронизации доступа процессов к буферу.

В подсистеме ввода-вывода для согласования скоростей обмена также широко используется буферизация данных в оперативной памяти. В тех специализированных операционных системах, в которых обеспечение высокой скорости ввода-вывода является первоочередной задачей (управление в реальном времени, услуги сетевой файловой службы и т. п.), большая часть оперативной памяти отводится не под коды прикладных программ, а под буферизацию данных. Однако буферизация только на основе оперативной памяти в подсистеме ввода-вывода оказывается недостаточной. Разница между скоростью обмена с оперативной памятью, куда процессы помещают данные для обработки, и скоростью работы внешнего устройства часто становится слишком значительной, чтобы в качестве временного буфера можно было бы использовать оперативную память - ее объема может просто не хватить. Для таких случаев необходимо предусмотреть особые меры, и часто в качестве буфера используется дисковый файл, называемый также спул-файлом (от spool - шпулька, тоже буфер, только для ниток). Типичный пример применения спулинга дает организация вывода данных на принтер.

111. Особенности важнейших задач подсистемы ввода-вывода.

1**. Организация параллельной работы УВВ и процессора.**

Каждое УВВ ВС снабжено специализированным блоком управления – контроллером, который взаимодействует с драйвером – системным программным модулем, предназначенным для управления данным устройством.

2. **Согласование скоростей обмена и кэширование данных**.

Согласование скоростей обычно достигается за счет буферизации данных в ОП и синхронизации доступа процессов к буферу

3. **Разделение УВВ и данных между процессами.**

УВВ могут предоставляться процессам в монопольное или совместное (разделяемое) использование. При этом ОС должна обеспечивать контроль доступа теми же способами, что и при доступе процессов к другим ресурсам ВС – путем проверки прав пользователя или группы, от имени которых действует процесс, на выполнение данной операции над УВВ.

4. **Обеспечение удобного логического интерфейса между УВВ и остальной частью ОС**.

Разнообразие УВВ делает особенно актуальной функцию ОС по созданию экранирующего логического интерфейса между периферийными устройствами и приложениями.

5. **Поддержка широкого спектра драйверов и простота включения новых**

Драйвер взаимодействует,

- с одной стороны, с модулями ядра ОС (модулями подсистемы ввода-вывода, системных вызовов, подсистем управления процессами и памятью);
- с другой стороны, с контроллерами УВВ.

Поэтому существуют два типа интерфейсов:

- интерфейс «драйвер-ядро» – он должен быть стандартизован в любом случае;
- интерфейс «драйвер-устройство» – его имеет смысл стандартизовать, если подсистема ввода-вывода не разрешает драйверу непосредственно взаимодействовать с аппаратурой контроллера, а делает это самостоятельно.

6. Динамическая загрузка и выгрузка драйверов.

Включение драйвера в состав модулей работающей ОС и его выключение «на ходу», то есть динамическая загрузка/выгрузка драйверов является важным свойством ОС, поскольку набор потенциально поддерживаемых данной ОС УВВ всегда делается существенно больше набора реальных устройств, которыми ОС должна управлять при ее установке на конкретном компьютере. При этом актуальна возможность динамически загружать в ОП требуемый драйвер без останова ОС и так же выгружать ненужный драйвер для экономии ОП.

7. Поддержка нескольких ФС.

Обычно большая часть системных и пользовательских данных хранится на дисках, делая дисковые накопители важнейшими УВВ. Данные на дисках организуются в различные ФС, а свойства отдельной ФС во многом определяют такие важнейшие свойства самой ОС, как отказоустойчивость, быстродействие, максимальный объем хранимых данных.

8. Поддержка синхронных и асинхронных операций ввода-вывода.

Операция ввода-вывода по отношению к запросившему ее программному модулю может выполняться, как и в случае ранее рассмотренных системных вызовов, синхронно (с приостановкой работы модуля и ожиданием ее завершения) или асинхронно (с продолжением работы модуля параллельно с операцией ввода-вывода). Причем, операция ввода-вывода может быть инициирована не только пользовательским процессом (когда она выполняется в рамках системного вызова), но и кодом ядра.

112. Обобщенная структура подсистемы ввода-вывода.

Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие УВВ(устройства ввода-вывода), и ФС(файловая система). К

подсистеме ввода-вывода можно условно отнести и диспетчер прерываний, обслуживающий не только ее модули, но и другие модули ОС.

Устройство ввода-вывода - служит для передачи данных между компьютером и внешним окружением. В число УВВ входят: внешняя (вторичная) память, терминал (монитор на основе электронно-лучевой трубки, жидкокристаллический, плазменный или иной, клавиатура, мышь, джойстик и другие устройства указания), дисководы для различных носителей, коммуникационное оборудование, оборудование мультимедиа, специальное оборудование (объекты управления, датчики, устройства поддержки среды виртуальной реальности и т.п.). Модуль ввода-вывода служит для передачи данных как от УВВ в процессор и память, так и обратном направлении. Для временного хранения данных в нем есть свои внутренние буферы.

Драйвер – специальная программа, которая анализирует состояние УВВ, передает данные, если надо преобразуя их, обрабатывает сбои УВВ, контролирует ошибки.

113. Организация и особенности менеджера ввода-вывода.

Верхний слой менеджера составляют системные вызовы ввода-вывода, которые принимают от пользовательских процессов запросы на ввод-вывод и переадресуют их отвечающим за определенный класс устройств модулям и драйверам, а также возвращают процессам результаты операций ввода-вывода.

Нижний слой менеджера реализует непосредственное взаимодействие с контроллерами внешних устройств, экранируя драйверы от особенностей аппаратной платформы компьютера — шины ввода-вывода, системы прерываний и т. п. Этот слой принимает от драйверов запросы на обмен данными с регистрами контроллеров в некоторой обобщенной форме с использованием независимых от шины ввода-вывода адресации и формата, а затем преобразует эти запросы в зависящий от аппаратной платформы формат

Важной функцией менеджера ввода-вывода является создание некоторой среды для остальных компонентов подсистемы, которая бы облегчала их взаимодействие друг с другом.

114. Организация и особенности многоуровневых драйверов.

- входит в состав ядра ОС, работая в привилегированном режиме;
- непосредственно управляет УВВ, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;
- обрабатывает прерывания от контроллера УВВ;
- предоставляет прикладному программисту удобный логический интерфейс для работы с некоторым УВВ, экранируя от него низкоуровневые детали управления им и организации его данных;
- взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур ввода-вывода,

которыми драйвер может пользоваться и т.п.

115.Назначение и функции классического драйвера

1. Обработка запросов записи-чтения от программного обеспечения управления устройствами. Постановка запросов в очередь.

2. Проверка входных параметров запросов и обработка ошибок.

3. Инициализация устройства и проверка статуса устройства.

4. Управление энергопотреблением устройства.

5. Регистрация событий в устройстве.

6. Выдача команд устройству и ожидание их выполнения, возможно, в блокированном состоянии, до поступления прерывания от устройства.

7. Проверка правильности завершения операции.

8. Передача запрошенных данных и статуса завершенной операции.

9. Обработка нового запроса при незавершенном предыдущем запросе (для реентерабельных драйверов).

116. Состав и назначение ФС, задачи ФС разных классов ОС.

Состав и назначение ФС(файловая система).

ФС – это часть ОС, включающая:

- совокупность всех файлов на диске или другом носителе для компьютера;
- наборы правил, конструкций и структур данных, используемых для хранения файлов и управления ими (каталоги файлов, дескрипторы файлов, таблицы распределения занятого и свободного пространства на диске). Эти наборы определяют конкретный тип ФС;
- комплекс системных программных средств, реализующих различные операции над файлами (создание, удаление, чтение, запись, именование, поиск и другие).

Задачи ФС разных классов ОС. Задачи, решаемые ФС, зависят от способа организации вычислительного процесса в ОС. Функции наиболее простых ФС в однопользовательских однопрограммных ОС (MS-DOS) нацелены на решение следующих задач:

- именование файлов;
- программный интерфейс для приложений;

- отображение логической модели ФС на физическую организацию хранилища данных;
- устойчивость ФС к сбоям питания, ошибкам аппаратных и программных средств.

117. Различные типы файлов, которые поддерживают ФС.

Типы файлов. ФС поддерживают несколько функционально различных типов файлов, в число которых входят:

- **обычные файлы (просто файлы)** – содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате выполнения системных или пользовательских программ. ОС обычно не ограничивает и не контролирует содержимое и структуру обычного файла, поскольку они определяются работающим с файлом приложением.
- **файлы-каталоги** – содержат системную справочную информацию о наборе файлов, сгруппированных пользователем или ОС по какому-либо признаку.
- **специальные файлы** – это фиктивные файлы, ассоциированные с УВВ, используемые для унификации механизма доступа к файлам и внешним устройствам. Они позволяют пользователю выполнять операции ввода-вывода с помощью обычных команд работы с файлами.
- **именованные конвейеры**. Конвейеры как средство межпроцессного обмена впервые появились в ОС UNIX. Конвейер (pipe) представляет собой буфер в ОП, поддерживающий очередь байтов по дисциплине FIFO.
- **отображаемые в память файлы.** Отображение файла в ВАП процесса применяется для упрощения программирования, позволяет работать с данными файла с помощью адресных указателей как с обычными переменными программы, без использования несколько громоздких файловых функций read/write (и явного описания необходимой области файла)

118. Имена файлов, иерархическая структура и монтирование ФС, логическая организация файла.

Иерархическая организация пространства имен позволяет облегчить эту работу за счет распределения файлов по группам (каталогам)
Граф, описывающий иерархию каталогов, может быть

- ✓ деревом, если файл может входить только в один каталог;
- ✓ сетью (с петлями/циклами), если файл может одновременно входить в несколько каталогов.

Имена файлов. Все типы файлов имеют различные по назначению символьные имена:

- простые (короткие), идентифицирующие файл в пределах каталога;

- составные (полные) – длинные, идентифицирующие файл в пределах дерева каталогов;
- относительные, идентифицирующие файл в пределах «текущего» каталога.

Монтирование.

Возможны два варианта реализации структур и именования файлов.

Первый вариант состоит в том, что на каждом устройстве файлы размещаются иерархически автономно, с собственным деревом каталогов.

Для однозначной идентификации файла пользователь должен перед составным символьным именем указывать идентификатор логического устройства (C:\primer\1\100\file17.doc).

Второй вариант представляет возможность пользователю самому объединять разрозненные иерархии файлов на различных устройствах (и описывающие их деревья каталогов) в одну иерархию и единое дерево каталогов. Такая операция называется монтированием

Логическая организация файла. В общем случае данные в файле имеют некую логическую структуру. Поддержка логической структуры может быть возложена на приложение или на ФС. В первом случае файл представляется ФС неструктурированной последовательностью байтов, а истинный формат известен только заинтересованному приложению.

Во втором случае ФС видит уже структурированный файл как порядоченную последовательность логических записей. ФС может использовать два способа доступа к логическим записям:

- последовательный – прохождение и поиск с начала;
- прямой – сразу по номеру записи.

119. Организация дисков, их секторов, блоков и кластеров, процедуры форматирования дисков, разделы и их свойства.

Диски, разделы, секторы, кластеры. Дисковые накопители являются основным типом устройств хранения файлов в современных ВС.

Жесткий диск состоит из одной или нескольких (пакета) стеклянных или металлических пластин, покрытых с одной или двух сторон магнитным материалом. На каждой стороне пластины размечены тонкие концентрические кольца – дорожки (tracks), на которых хранятся данные. Совокупность всех дорожек одного радиуса на всех поверхностях всех пластин пакета называется цилиндром (cylinder). Каждая дорожка разбивается на фрагменты – секторы (sectors) или блоки (blocks). Все дорожки имеют равное число секторов, куда можно записать равное число байтов. Сектор имеет фиксированный для

конкретной системы размер, равный 2k (обычно 512 байтов). Сектор – наименьшая адресуемая единица обмена данными дискового устройства с ОП.

Форматирование жёсткого диска включает в себя три этапа:

1. **Низкоуровневое форматирование**. Это базовая разметка области хранения данных, которая выполняется на заводе-изготовителе в качестве одной из заключительных операций изготовления устройства хранения данных. При этом процессе в области хранения данных создаются физические структуры: треки — *tracks* (дорожки), секторы, при необходимости записывается программная управляющая информация. Впоследствии в подавляющем большинстве случаев эта разметка остаётся неизменной за все время существования носителя. Большинство программных утилит с заявленной авторами возможностью низкоуровневого форматирования на самом деле, в лучшем случае, перезаписывают только управляющую информацию.
2. **Разбиение на разделы**. Этот процесс разбивает объём винчестера на логические диски .Это осуществляется с помощью встроенных служб самой операционной системы или соответствующими утилитами сторонних производителей (см. Программы для работы с разделами); метод разбиения существенно зависит от типа операционной системы. Этот шаг принципиально необязателен, но в виду очень больших объемов современных жестких дисков их разбиение на логические разделы обычно осуществляется.
3. **Высокоуровневое форматирование**. Этот процесс записывает (формирует) логические структуры, ответственные за правильное хранение файлов (файловые таблицы), а также, в некоторых случаях, загрузочные файлы для разделов, имеющих статус активных. Это форматирование можно разделить на два вида: быстрое и полное. При быстром форматировании перезаписывается лишь таблица файловой системы, при полном — сначала производится верификация (проверка) физической поверхности носителя, при необходимости исправляются поврежденные сектора, то есть участки оптической поверхности, имеющие физические повреждения (маркируются как неисправные, что исключает в последующем запись в них информации), а уже потом производится запись таблицы файловой системы.

120. Физическая организация (размещение) и адресация файла

Важным компонентом физической организации ФС является физическая организация файла – способ размещения файла на диске. Основными критериями эффективности физической организации файлов являются:

- скорость доступа к данным;
- объем адресной информации файла;
- степень фрагментированности дискового пространства;
- максимально возможный размер файла.

Непрерывное размещение – простейший вариант физической организации, когда файлу предоставляется последовательность смежных кластеров. Его достоинства: высокая скорость доступа, минимум объема адресной информации, отсутствие ограничения максимального размера.

Недостатки: сложность поддержки увеличения размера, подверженность фрагментации. используются методы размещения файла в несмежных областях диска:

- в виде связанного списка кластеров, когда в начале каждого кластера содержится указатель на следующий кластер.

Достоинства:

адресная информация минимальна (адрес первого кластера файла), удобство увеличения размера файла, отсутствие фрагментации.

Недостатки:

последовательный доступ; из-за того, что одно слово израсходовано на номер следующего кластера, объем данных кластера уже не равен 2k, а многие программы читают данные именно кластерами размера 2k;

- с использованием связанного списка индексов. Это модификация предыдущего метода, когда номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связывается некоторый элемент – индекс. Индексы располагаются в отдельной области диска.
- простое перечисление номеров кластеров файла. Этот перечень и служит адресом файла. Недостаток: длина адреса зависит от размера файла.Достоинства: высокая скорость доступа к любому кластеру ввиду прямой адресации, отсутствие фрагментации.