МИНОБРНАУКИ РФ
Федеральное агентство по образованию

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.Э.БАУМАНА
(МГТУ ИМ. Н.Э.БАУМАНА)

Факультет Информатики и Систем Управления (ИУ)
Кафедра «Компьютерные системы и сети» (ИУ-6)

Беккер М.В.

Операционные системы

Лекции

Москва, 2017 г.

СОДЕРЖАНИЕ

ОБЩАЯ ИНФОРМАЦИЯ

Требования:

1) 3 модуля ⇒ 3 РК по ним:
   - 1 модуль – 20 баллов, РК на 5 неделе;
   - 2 модуль – 30 баллов, РК на 12 неделе;
   - 3 модуль – 20 баллов, РК на 16 неделе.
2) 9 лабораторных работ, за которые добавляются бонусные баллы;
3) Обязательное ДЗ – реферат, который сдается на 14-ой неделе. За него дается 10 баллов (общий максимум) или больше – если отсутствуют ссылки на любую литературу и все исследование произведено самим студентом. Минимальный процент уникальности – 70 %.

Всего на курс отводится 30 часов на лекции и 34 часа на лабораторные работы.

Автомат:

1) Автомата на УДОВЛ нет;
2) Автомат на ХОР ставится без всяких вопросов – при условии набора нужного количества баллов;
3) Автомат на ОТЛ ставится с минисобеседованием (вопрос).

Основные разделы (остальные входят в них, рассматриваются дополнительно):

1) Управление процессами;
2) Управление памятью;
3) Файловая система.

Рекомендованная литература:

1) Иртегов Д. Введение в операционные системы (2-е издание – BHV Русская редакция, 2011 – 1040 с.);
2) Танненбаум Э. Современные операционные системы (3-е издание – СПб, 2010 – 1116 с.).

Примерные темы рефератов:

1) Сравнительный анализ операционных систем (... с ... / одного семейства / в целом);
2) Тоже самое для файловой системы;
3) Эволюция ОС (ни о чем);
4) Управление процессами и потоками (планирование, синхронизация, распараллеливание);
5) Методы управления ОЗУ в современных ОС;
6) Логическая организация файловых систем (ReFS);
7) Физическая модель ФС;
8) Защита ресурсов в различных видах ОС (ФС, доступ в целом, сеть);
9) ОСРВ, мобильные ОС;
10) Микроядерные ОС;
11) Совместимость ОС между собой;
12) Основные свойства ОС – сравнительный анализ между разными ОС;
13) Семантика разделения файлов.

Я взял тему: **«Микроядро Mach и ОС на ее основе»**.

Примечание Обозначаю ***пример*** символом #, ***и т.д.*** как ***etc.***, ***оперативную память*** как ***RAM***, постоянную память как ***ROM***, ***процессор*** как ***ЦПУ***, ***операционную систему*** как ***ОС***.

1. ПЕРВАЯ ЛЕКЦИЯ

1.1. Свойства ОС

Любая ОС должна обладать следующим набором свойств:

1) **Надежность.**

   ОС должна быть надежна, как и аппаратура, на которой она работает. Что конкретно подразумевается – должны быть средства определения, диагностирования и восстановления после большинства ошибок;

2) **Защита.**

   # Защита от взаимного влияния задач пользователей друг на друга; минимизация порчи программ и данных;

3) **Предсказуемость.**

   Реакция системы должна быть та, которую от нее ожидает пользователь и не варьироваться слишком сильно;

4) **Удобство.**

   Тут все очевидно – дружелюбный UI, обеспечение работы пользователя, проектирование с учетом фактора человеческой психологии;

5) **Эффективность.**

   # Эффективность при распределении и использовании ресурсов (RAM, ROM, etc.);

6) **Общие системные услуги.**

   Основная идея: услуги должны быть такими, чтобы пользователь при решении задач как можно реже обращался к доп. возможностям ОС;

7) **Гибкость.**

   # При установке системы; настройка системных операций;

8) **Масштабируемость.**

   Т.е. возможность добавления новых средств в ОС;

9) **Прозрачность (ясность).**

   Т.е. пользователь имеет возможность знать об ОС все, что он пожелает - в том числе возможность спуститься на уровень ядра.

1.2. Основные понятия и определения

Операционная система – организованный программный комплекс и набор средств, позволяющих упростить программирование, отладку и сопровождение программ. В целом, ОС представляет собой интерфейс между пользователем и аппаратной составляющей ЭВМ.

Процесс – совокупность последовательных действий, необходимых для достижения какого-либо результата вычислительной системы. В современных ОС процесс реализуется как динамический объект, который имеет множество полей и методов (*атрибутов* и *сервисов*), может находиться в различных состояниях.

Выполнять действия по отношению к другим процессам объекта, занимать или освобождать необходимые ресурсы, etc.

Процессы могут быть последовательными относительно друг друга, параллельными или комбинированными. Процесс еще можно назвать логической единицей работы операционной системы.

Поток (нить) – сущность внутри процесса. Она отображает одну из, возможно, многих подзадач процесса.

Многозадачность – способ организации вычислительного процесса, при котором различные потоки совместно используют ЦПУ.

Кооперативная многозадачность – многозадачность, при которой потоками пользуется не только ЦПУ, но и еще другие ресурсы: RAM, файлы, etc.

Многопоточность – поддержка нескольких потоков внутри одного процесса. Реально это достигается, если число процессоров (*ядер*) больше или равно числу потоков.

Мультипроцессорная обработка – исполнение одного и того же кода ОС различными процессорами как на однопроцессорных, так и на многопроцессорных ЭВМ. При этом возможно мультипрограммирование и на каждом из процессоров может попеременно выполняться закрепленный набор потоков.

Вытесняющая многозадачность (дисциплина обслуживания) – решение о переключении процессора с одного потока на другой принимает ОС (в невытесняющих поток по собственной инициативе передает управление ОС).

Ресурсы – различные средства и возможности, необходимые для организации (порождения) и поддержки процесса. ОС должна обеспечить эффективный и бесконфликтный способ распределения (разделения) ресурсов между процессами.

В итоге, ресурсы могут быть:

1) Физическими (реально существуют) и виртуальными (не существуют в том виде, в котором проявляют себя пользователю, но построение производится на базе физического; по существу, это модель некоторого физического ресурса – # реестр);

2) Пассивными и активными (могут выполнять действия в отношении к другим ресурсам);

3) Временными и постоянными.

Категории ресурсов:

1) Процессорное время;

2) Память (оперативная, внешняя, виртуальная);

3) Внешние устройства;

4) Математическое обеспечение.

2. ВТОРАЯ ЛЕКЦИЯ

2.1. Эволюция ОС

Первый этап эволюции – в первых ламповых ЭВМ (середина 1940-х годов) программирование осуществлялось в машинных кодах. Задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Были только библиотеки математических и служебных программ (утилит). В каждый момент времени работы ЭВМ выполнялась только одна задача.

В ЭВМ 2-го поколения (середина 1950-х годов), базировавшихся на полупроводниковых транзисторах, появились первых компиляторы и операционные системы пакетной обработки (**пакет** – совокупность отдельных программ и данных пользователя, разделенных специальными метками на магнитной ленте). Здесь уменьшается влияние пользователя, появляется оператор, который формирует пакет и появляется программа, которая считывает последовательно пакеты, запускает их, отслеживает аварийные ситуации, фиксирует время выполнения, etc. Такую программу, явно управляющая, ее можно назвать простейшей ОС.

Второй этап эволюции – ОС мультипрограммной пакетной обработки. Здесь уже обеспечивается эффективное использование ресурсов несколькими пользователями (в оперативной памяти несколько пользовательских программ, время процессора разделяется, etc.). На этом этапе одновременно могут работать ЦПУ, ОЗУ, каналы, внешние устройства.

Третий этап эволюции – ОС с разделением времени. Основной целью было обслужить каждого пользователя и обеспечить допустимое время реакции ЭВМ на его запросы. В схему по мультиплексированию ЦПУ вводится *детерминизм*, т.е. каждой программе отводится свой интервал (квант) времени, причем, в случае того, что она не успела уложиться в свой интервал времени, она вновь встает в очередь.

Четвертый этап эволюции – многопоточные и многопроцессорные ОС.

Пятый этап эволюции – микроядерные ОС. В микроядре изолирована вся машино-зависимая часть.

2.2. Основные функции и типы ОС

В результате, основными функциями ОС являются:

1) Управление процессами (задачи планирования, синхронизации, взаимодействия с процессами, etc.) **[Главная]**;

2) Управление ресурсами (организация доступа ЦПУ, управление памятью) **[Главная]**;

3) Управление внешними устройствами;

4) Защита.

С точки зрения пользователя, число функций ОС может быть больше:

1) Поддержка операционного окружения пользовательских задач;

2) Обеспечение совместимости с другими ОС;

3) Защита и безопасность информации.

Всякая ОС является ОС одной из следующих типов:

1) **ОС ЭВМ общего назначения.**

   Для широкого круга пользователей;

2) **ОСРВ.**

   Управление датчиками, много внешних устройств; широкий спектр средств ввода-вывода; упрощенный алгоритм обработки; высокая надежность.

3) **ОС портативных ЭВМ.**

4) **ОС специального назначения.**

3. ТРЕТЬЯ ЛЕКЦИЯ

3.1. Основные типы структур ОС (эволюции)

3.1.1. Монолитная ОС

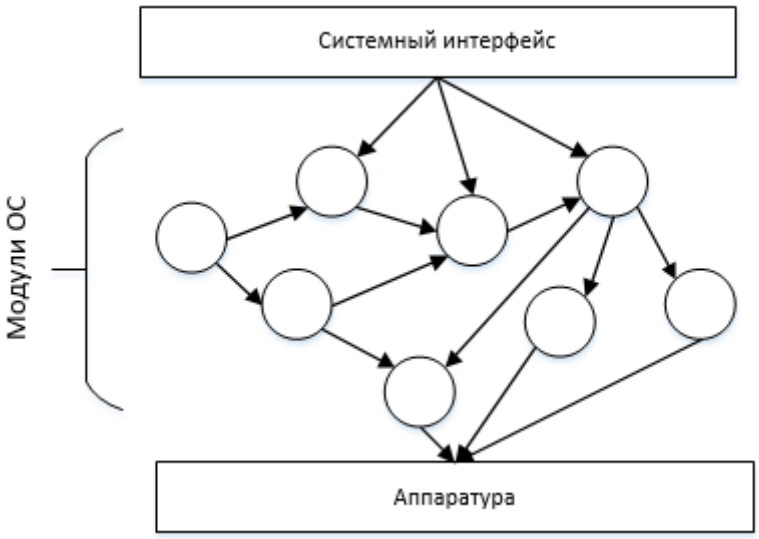

Рис. 1. Схематическая модель построения монолитной ОС

ОС – виде набора процедур, где каждая из процедур может вызвать любую другую в произвольном порядке (отсутствуют правила вертикального управления). Для построения монолитной ОС необходимо: скомпилировать все отдельные процедуры и связать их в единый объектный файл (с помощью компоновщика). Так были созданы первые версии UNIX и NetWare.

3.1.2. Структурированная монолитная ОС

Т.е. монолитная ОС отчасти структурирована (к.о.). При обращении к системным вызовам параметры помещаются в строго определенные места, например, регистры или стек, а, затем, выполняется специальные команды прерывания, известные как вызов ядра (супервизоров). Эта команда переключает машину из *режима пользователя* в *режим ядра*, затем проверяются параметры вызова, индексируется таблица ссылок на процедуры и вызывается соответствующая процедура. Для каждого системного вызова имеется одна системная процедура.

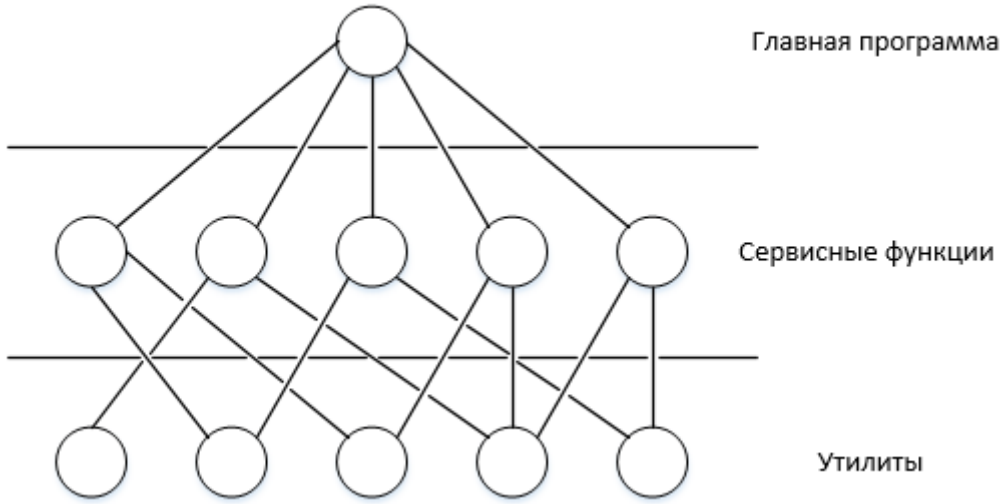

Рис. 2. Схематическая модель построения структурированной монолитной ОС

Утилиты – программы, обслуживающие сервисные процедуры, причем одна из них может обслуживать несколько процедур;

3.1.3. Многоуровневые иерархические системы

В 1968 году, Дейкстра предложил простую пакетную ОС с 6-ю уровнями:

- **Нулевой уровень** – отвечал за распределение времени ЦПУ, переключал процессы по прерыванию или по истечению кванта времени;
- **Первый уровень** – выполнял функции виртуальной оперативной памяти;
- **Второй уровень** – управлял связью между консолью оператора и процессами;
- **Третий уровень** – управлял устройствами ввода-вывода и буферизировал потоки;
- **Четвертый уровень** – пользовательские программы;
- **Пятый уровень** – процесс системного оператора.

Ограничение данной ОС: каждый из уровней может взаимодействовать только с непосредственно примыкающим к нему;

3.1.4. Микроядерная ОС

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

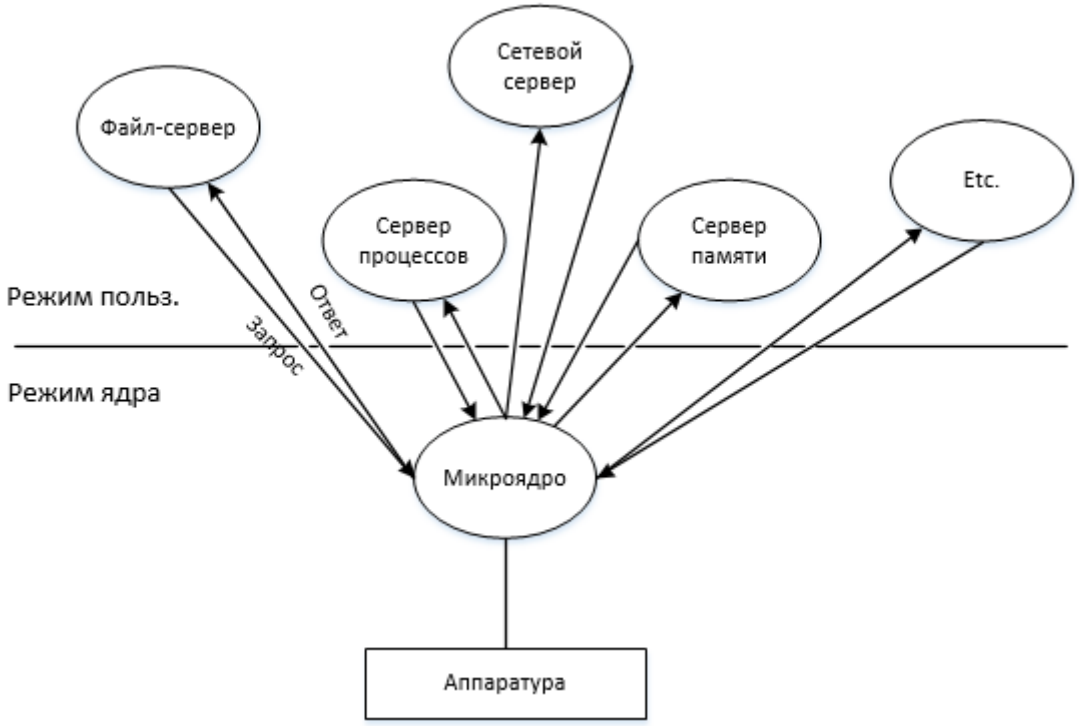

Рис. 3. Схематическая модель построения микроядерной ОС

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

В Windows NT в некоторых случаях для оптимизации используется разделенная память – это дополнительные расходы в ресурсах.

3.2. Структура ОС на примере DOS 6.2

BIOS – ***B****asic* ***I****nput-****O****utput* ***S****ystem - Базовая Система Ввода-Вывода.* Находится в ROM.

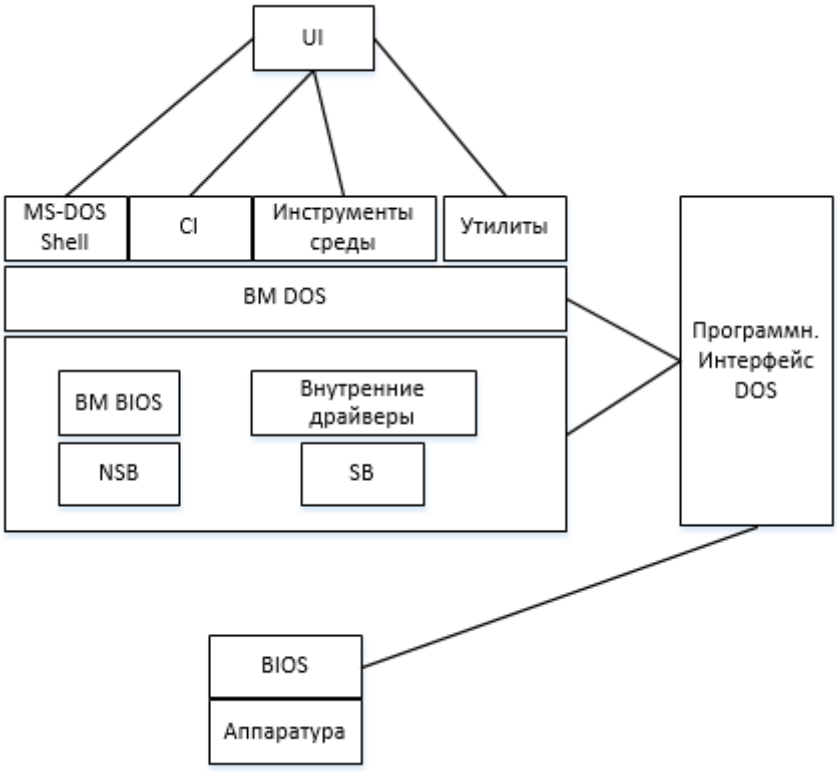

Рис. 4. Структура ОС DOS 6.2

Функции BIOS:

- Функция ***POST*** – ***P****ower-****O****n* ***S****elf-****T****esting* – самотестирование компьютера при включении;
- Загрузка блока начальной загрузки (знаю, что тавтология) DOS с диска в оперативную память;
- Обслуживание системных вызовов – используется механизм прерываний, т.е. работа машины приостанавливается одним из сигналов с целью срочной обработки. Каждое прерывание имеет свой уникальный номер и с ним может быть связана определенная подпрограмма.

  Прерывания можно разделить на три группы:

  1) **Аппаратные.**

     Возникают при падения напряжения питания, нажатия кнопок и клавиш, при импульсах с счетчиков времени, etc., которые идут с устройств нижнего уровня;

  2) **Логические (процессорные).**

     Возникают при нестандартных ситуациях.

     # Деление на ноль, переполнение регистров, появление точки останова, etc.;

3) **Программные.**

   Они вырабатываются, когда одна программа захочет получить сервис со стороны другой программы, причем, этот сервис может быть связан и с аппаратными средствами.

Прерывания нижнего уровня имеют следующие номера: [0..31], [0..1F]. Прерывания высокого уровня имеют следующие номера: [32-63], [20-3F]. Их обслуживание возлагается на другие модули DOS.

В итоге, BIOS содержит:

1) Драйверы стандартных периферийных устройств;
2) Тестовые программы аппаратуры;
3) Программы начальной загрузки.

- Осуществляет инициализацию векторов нижнего прерывания и считывает в память NSB (Non System Bootstrap) – стартовый сектор физического жесткого диска.

NSB считывает в память и запускает SB. SB – стартовый сектор каждого логического диска.

Функции SB:

- Считывает в память EMBIOS;
- Считывает в память BMDOS;
- Передает управление EMBIOS.

EMBIOS осуществляет:

- Осуществляет определения состояния оборудования;
- Конфигурирует DOS по файлу CONFIG.SYS;
- Обеспечивает инициализацию и переустановку некоторых векторов прерываний нижнего уровня;
- Запускает BMDOS;

BMDOS осуществляет основные функции ОС – управление процессами и ресурсами. Его основу составляют обработчики прерываний верхнего уровня. BMDOS осуществляет считывание в память и запуск CI (Command Interpreter) (находится в файле COMMAND.COM). Он отвечает за поддержку GUI и осуществляет выполнение файла AUTOEXEC.BAT.

4. ЧЕТВЕРТАЯ ЛЕКЦИЯ

4.1. Структура операционной системы Windows

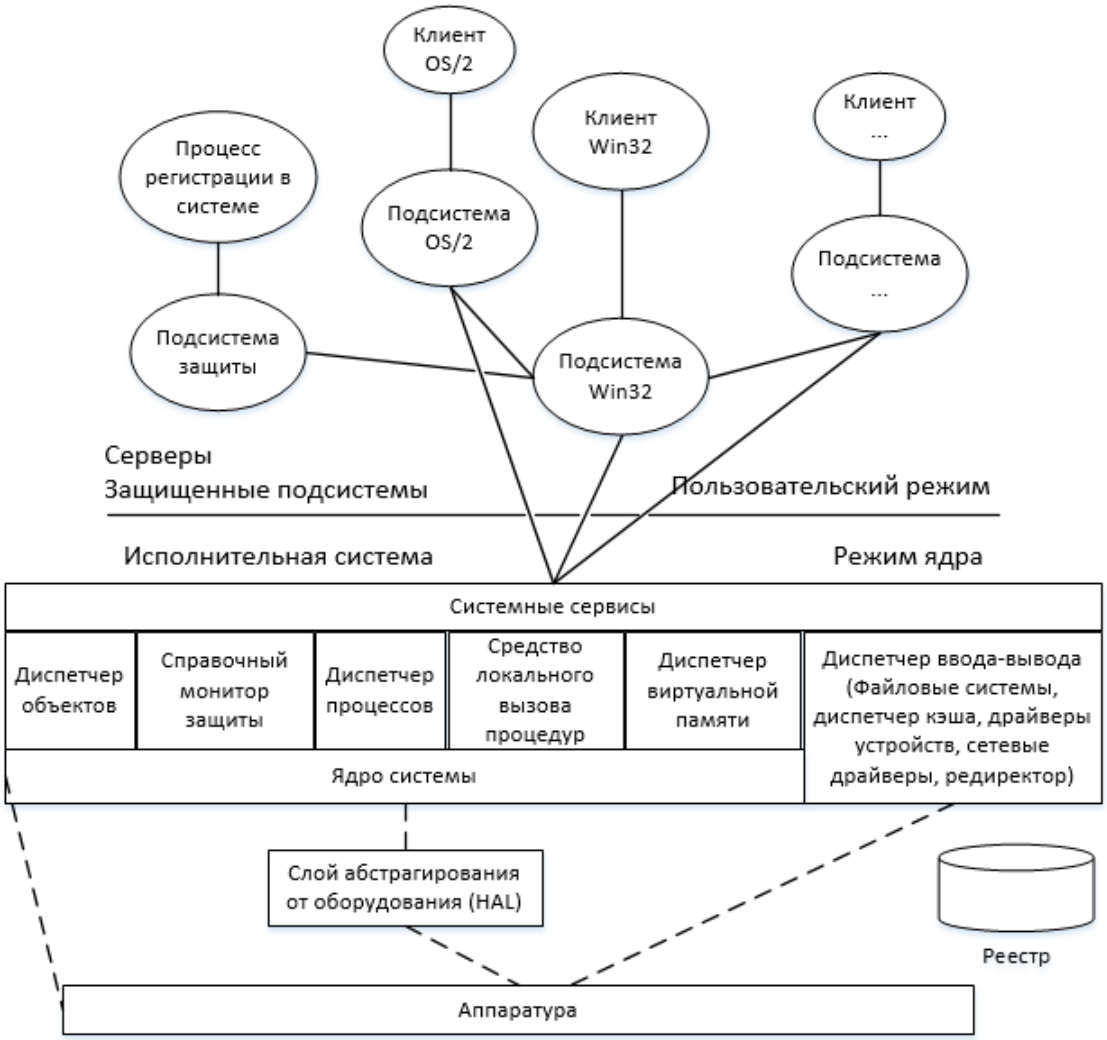

Рис. 5. Структура ОС Windows

Структуру Windows можно разбить на две части:

1) **Защищенные подсистемы (серверы)** – каждая из серверов является отдельным процессом, память которого защищена от других процессов с помощью системы виртуальной памяти исполнительной системы. Они предоставляют исполнительной системе пользовательские и программные интерфейсы, обеспечивают среды для выполнения приложений различных типов.

   API – ***A****pplication* ***P****rogramming* ***I****nterface* – представляет собой набор процедур, который вызывается прикладной программой для выполнения низкоуровневых операций. API реализуется на отдельном сервере для Win32. Также имеются старые системы (для 16-ти разрядной Windows, MS-DOS, etc.). Выделение в отдельный сервер позволило устранить конфликты и дублирование в исполнительной системе. Самая главная система – Win32, т.к. предоставляет API для 32-разрядной Windows, а также реализует UI и управляет вводом-выводом.

Остальные подсистемы имеют свой API, но используют для ввода и отображения результатов общий интерфейс Win32.

Подсистема защиты регистрирует правила контроля доступом на локальной ЭВМ, а именно: ведет БД учетных записей пользователей, etc.;

2) **Исполнительная система** – сама по себе законченная ОС и выполняет функции низкого уровня. Имеет два вида функций:

   - Системные сервисы;
   - Внутренние процедуры;

   Диспетчер объектов (наверное, также инспектор объектов) создает, поддерживает и уничтожает объекты, которые предоставляют системные ресурсы (процессы).

   Справочный монитор защиты оберегает ресурсы ОС, осуществляет защиту объектов и аудит (возможность обнаружения и регистрации важных событий, относящихся к защите) во время выполнения.

   Диспетчер процессов создает и завершает процессы и потоки и выдает о них информацию.

   Средство локального вызова процедур (**LPC** – ***L**ocal **P**rocedure **C**all*) передает сообщения между клиентскими и серверными процессами, но расположенными только на одной и той же ЭВМ.

   Диспетчер виртуальной памяти – реализует виртуальную память, управляет ей во время работы и предоставляет каждому процессу собственное адресное пространство. Также отвечает за подкачку (свопинг) страниц (сегментов) в память, виртуальные машины (в частности – **VDM** – ***V**irtual **D**OS-**M**achine*), etc.

   Ядро системы отвечает за обработку прерываний, направляет потоки на выполнение, осуществляет межпроцессорную синхронизацию, а также скрывает процессорные различия от остальной части ОС, etc. Само ядро имеет три компонента:

   - USER – управляет вводом с клавиатуры, мыши, и другие устройства, а также отвечает за окна, меню, etc.;
   - GDI;
   - Kernel (само ядро) – обеспечивает базовые функции ОС.

   Диспетчеры ввода-вывода *(ФС – смотри отдельный раздел).*

   Диспетчер кэша повышает производительность файлового ввода-вывода. Там используют средства подкачки страниц из виртуальной памяти с целью автоматической записи информации на диск в фоновом режиме (асинхронная запись на диск).

   Сетевой редиректор – отвечает за запросы ввода-вывода для удаленных (сетевых) файлов, а также именованных каналов (почтовых ящиков) и отвечает за пересылку запросов к сетевому серверу на другую ЭВМ.

   Драйверы устройств (библиотеки устройств) – применяется архитектура «универсального драйвера\минидрайвера». Универсальный драйвер включает общую часть кода

для класса устройств: принтеров, модемов, etc. Минидрайвер более простой и содержит инструкции для конкретного устройства.

Слой абстрагирования от оборудования (**HAL** – ***H****ardware* ***A****bstraction* ***L****ayer*) – по-сути, динамически подключаемая библиотека (.DLL). Этот слой изолирует исполнительную систему от особенностей аппаратных платформ различных производителей. Реализует функции: обеспечивает абстракции интерфейса ввода-вывода (API), обеспечивает абстракцией контроллера прерываний, аппаратных кэшей, механизмов межпроцессорных коммуникаций, etc.

Реестр – центральная информационная БД в ОС Windows, появившаяся в Windows 95. Появление реестра позволило упростить структуру ОС, т.к. пропадает необходимость в AUTOEXEC.BAT, CONFIG.SYS и INI-фалов. Осуществляет централизованное хранение информации (см. методу) – профили всех пользователей, данные о программах и типах документов, значения свойств для папок и значков программ, конфигурация оборудования, данные об использованных портах, etc. В целом, реестр имеет логически древовидную структуру, состоящую из разделов, подразделов, кустов и записей реестра.

4.2. Виртуальные машины

Виртуальная машина может выполнять:

1) Машино-независимый код (Byte-код, p-код, etc.);
2) Машинный код реального ЦПУ;
3) Эмулировать работу отдельных компонентов Hardware;
4) Работу реальной ЭВМ в целом;
5) Совместимость старых приложений (см. рис. 7, 8);

Диспетчер виртуальной памяти гарантирует, что физическая память не будет пересекаться с всеми VDM и другими процессами:

Рис. 6. Изоляция виртуальных машин

Совместимость старых приложений достигается путем следующей организации адресации:

Рис. 7. Совместимость DOS-приложений

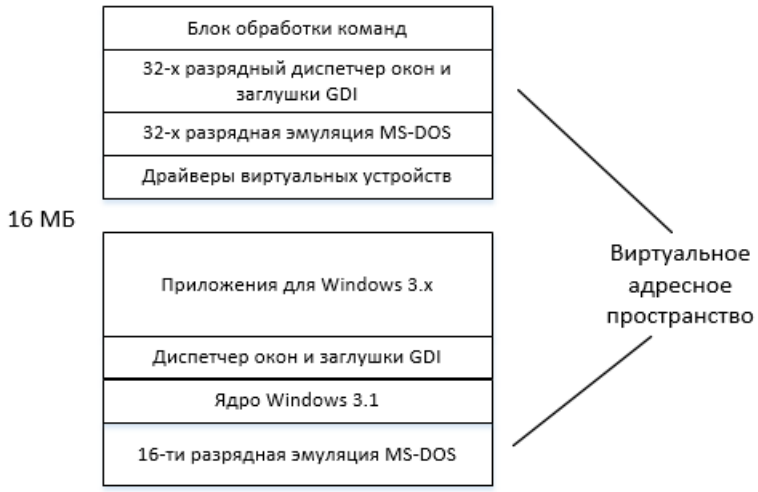

Рис. 8. Совместимость Windows 3.x-приложений

Виртуальная ОС не является полноценной ОС, а является уже приложением хостовой ОС. В сущности, VDM – это виртуальная ОС MS-DOS, выполняющаяся на виртуальной ЭВМ с ЦПУ архитектуры Intel-x86. Диспетчер обеспечивает использование всеми VDM одной копии 32-разрядного кода. Приложение MS-DOS не является многозадачным, но Windows работает с потоками DOS также, как и с другими.

5. ПЯТАЯ ЛЕКЦИЯ

5.1. Управление процессами и задачами

Процессы – один из способов управления программами в ходе их выполнения. Для выполнения процессов требуются различного рода ресурсов. Процессы могут быть различными – # пользовательский процесс создается, когда начинается выполнение задания и уничтожается, когда выполнения задания прекращается.

Процесс, как логическая единица, предполагает два аспекта:

1) Выполняет операции;

2) Является носителем данных.

Т.е. процессу присущи две части:

1) **Программа, по которой он будет выполняться в активном состоянии**;

2) **Дескриптор** – информационная структура, в которой сосредоточена управляющая информация. Эта информация необходима для планирования и управления процессом.

   # Очереди процессов представляет собой дескрипторы отдельных процессов, объединенных в список.

Классификация процессов на примере GNU/Linux:

1) **Системные процессы** – являются частью ядра и всегда находятся в RAM. Такие процессы не имеют соответствующих им программ в виде исполняемых файлов. Они запускаются особым образом при инициализации ядра ОС.

   # Системными процессами являются:

   - shed – swap-диспетчер;
   - vhand – диспетчер страничного замещения;
   - bdflush – диспетчер буферного кэша;
   - rmadaemon – диспетчер памяти ядра;
   - init – процесс-прародитель всех остальных процессов.

2) **Демоны (DAEMON)** – не интерактивные процессы, которые запускаются путем загрузки в память соответствующих им программ (исполняемых файлов), но они не связаны с пользовательскими – что-то вроде утилит. Выполняются в фоновом режиме. Обычно демоны запускаются при инициализации системы, но после инициализации ядра. Демоны обеспечивают работу различных подсистем *nix-like систем:

   # Систем терминального доступа, систем печати, сетевого доступа, услуг, etc.

   Демоны не связаны ни с одним пользовательским сеансом работы и, поэтому, не могут удаляться непосредственно пользователем;

3) **Прикладные** – все остальные.

   # Порожденные во время пользовательского сеанса работы.

Контекст процессов – информация о процессе, необходимая ему тогда, когда он находится непосредственно в активном состоянии.

По функциональному назначению, можно выделить следующие группы информации дескриптора:

1) Информация по идентификации;
2) Информация о ресурсах, которые ему требуются;
3) Информация о состоянии процесса – позволяет определить текущее состояние и возможность перехода в следующее (см. граф существования процесса и граф состояния);
4) Информация о родственных связях – используется для корректного завершения процесса;
5) Информация, необходимая для учета и планирования - содержит ссылки на средства синхронизации.

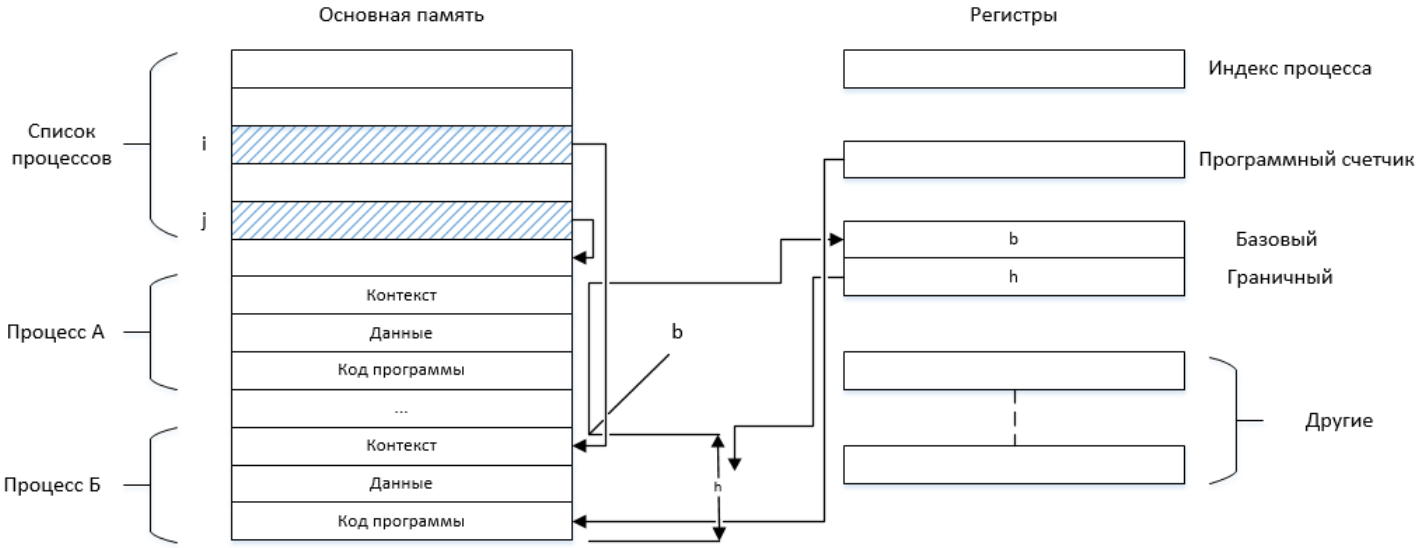

Рис. 9. Взаимодействия процессов

На рис. 9 отображены два процесса – А и Б, находящиеся в разных областях RAM. Каждый процесс заносится в список, в котором имеются указатели на области памяти процесса, где находится код программы, данные и информация о состоянии процесса.

Программный счетчик указывает на очередную инструкцию, которую нужно выполнить. В регистре индекса содержится индекс, выполняющийся в данный момент времени процесса. Базовый и граничные регистры задают область памяти, занимаемую процессом.

5.2. Граф существования процесса

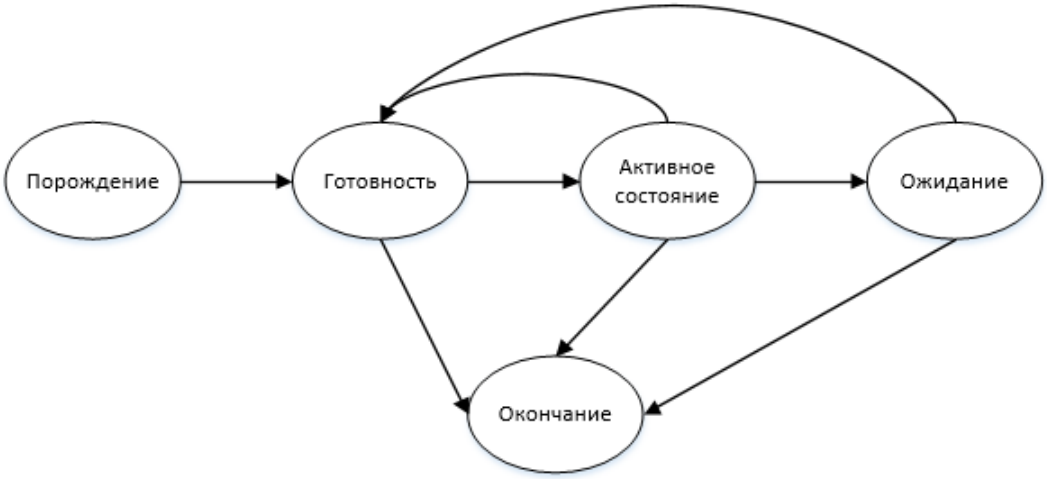

Рис. 10. Граф существования процесса

Процесс порождения – подготавливаются все условия для выполнения (читай, конструктор).

Готовность – предоставлены все ресурсы, кроме ресурсов ЦПУ и процесс ждет, когда планировщик (диспетчер) выберет его.

Активное состояние – процесс обладает всеми ресурсами и непосредственно используют ЦПУ.

Ожидание – процесс может быть прерван или приостановлен по нескольким причинам, которые можно разбить на 2 вида:

1) Проблема не в самом процессе – насильственное вытеснение по окончанию кванта времени, поступил процесс с более высоким приоритетом, необходимость синхронизации между процессами, etc.;

2) Проблема в процесса – попытка получить ресурс или отказаться от него, при порождении или уничтожении или действий к другим процессам, прерывания – арифметическое переполнение, обращение к защищенной области памяти, etc.

Выделяют отдельные состояния:

1) Готовый к выполнению (загруженный);

2) Готовый к выполнению (выгруженный);

3) Спящий (загруженный);

4) Спящий (выгруженный).

6. ШЕСТАЯ ЛЕКЦИЯ

6.1. Планирование и диспетчеризация процессов

В мультипрограммной ОС на ресурсы могут претендовать сразу несколько процессов (пользователей), поэтому необходимо осуществлять планирование.

Планирование процессов – управление распределением ресурсов вычислительной системы между различными процессами путем передачи управления согласно определенной стратегии.

Планировщик – набор функциональных модулей, отвечающих за операции управления. Он отвечает за постановку процесса в очередь, управляет структурой этой очереди.

Диспетчеризация – выбор процесса и передача ему управления.

Современные ОС имеют двухуровневую схему управления (планировщика):

1) Долгосрочное планирование (планировщик заданий) – ВЕРХНИЙ УРОВЕНЬ.

   Действия, редкие в системе, но требующие больших ресурсных затрат. Процесс рассматривается, как совокупность состояний по исполнению программы на виртуальной машины.

   # Состояние порождения – планировщик создает виртуальную машину;

2) Краткосрочное планирование (супервизор задач) – НИЖНИЙ УРОВЕНЬ.

   Моделируется на ЦПУ деятельность виртуального процесса, а состояние активности определяет исполнение работы на виртуальном процессоре. Заявка на порождение такой работы формируется на верхнем уровне.

   Доступ любого задания ЦПУ осуществляется через системные программы планировщика-диспетчера.

Можно выделить 2 основных стратегии построения планировщиков:

1) **Единый главный планировщик** – используется для всех заданий, находится обычно в ядре ОС. Основная задача – осуществляет разделение времени процессов;

2) **Разделенный планировщик** – планировочный модуль помещается в адресную часть каждой программы. Процесс может осуществить подпрограммный вызов для постановки самого себя в очередь на исполнение.

Алгоритмы тесно связаны с дисциплиной обслуживания очереди. Разберем некоторые основные (классические, широко используемые и простые) из этих дисциплин:

1) **Дисциплина обслуживания в порядке поступления – FIFO – очередь.**

Рис. 11. Пример FIFO

Она обеспечивает минимизацию дисперсии и математического ожидания;

2) **Дисциплина обслуживания в порядке, обратном порядке поступления – LIFO – стек.**

Рис. 12. Пример LIFO

Нужна для управления программами в ходе их выполнения.
Общим для FIFO и LIFO является то, что среднее время ожидания в очереди одинаково – не зависит от характеристик процессов.

Если одни процессы предполагают длительное использование ресурсов, а другие – наоборот, то что оба типа процессов будут в очереди ожидать, в среднем, одинаково.

Типы алгоритмов:

- **Циклическое планирование** – процессы выбираются из очереди и выполняются по порядку. Приоритет явно не задается и определяется линейным положением в очереди.

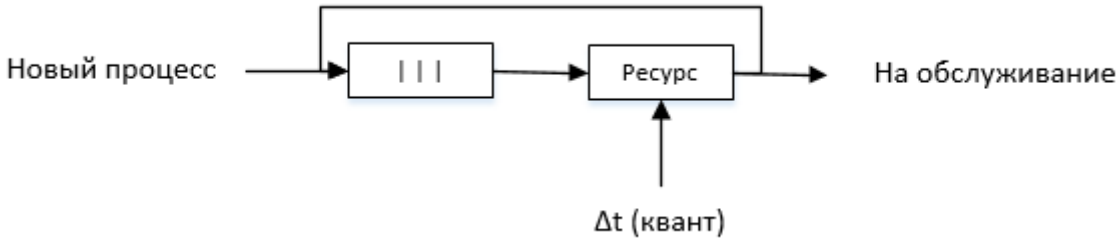

Рис. 13. Пример структуры циклического планирования

Недостаток – один процесс может занимать ЦПУ длительное время в ущерб другим процессам. Для решения этой проблемы вводится временной интервал (квант), по истечению которого процесс прерывается, обрабатывается и помещается в конец очереди. Здесь происходит дискриминация длинных запросов – т.е. короткие выполняются быстрее. Данный алгоритм используется в различных вариантах и ОС с разделением времени;

- **Приоритетное планирование.**

Приоритет – число, характеризующее степень привилегированности процесса при использовании ресурса. Значение может быть больше, меньше нуля, целым или дробным. Оно определяет положение каждого процесса относительно других в очереди на исполнение. Процесс с самым низким приоритетом называют холостым. Приоритеты могут быть разбиты на группы еще на этапе проектирования. Количество групп выбирается таким образом, чтобы во время обработки не происходило скопление процессов в отдельных группах.

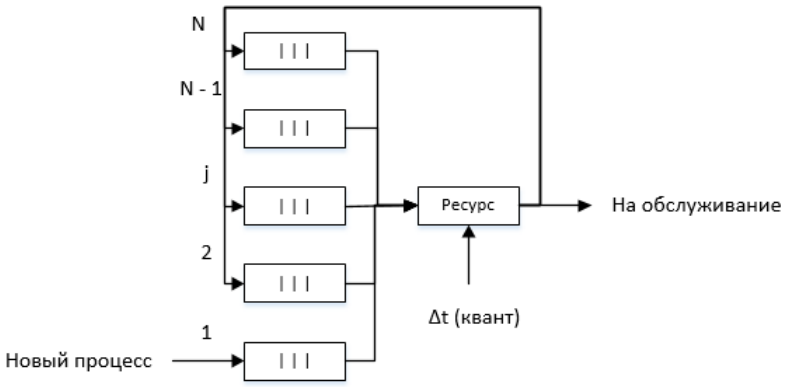

Рис. 14. Пример структуры приоритетного планирования

Число приоритетов от 0 до 255.

В целом, выделяют приоритетное планирование:

– **Статическое** – процессы при создании могут быть разделены на группы несколькими способами.

Назначение приоритеты исходя из запросов на ресурсы; согласно приоритету программы, которой принадлежит данный процесс; оценкой времени выполнения всей программы; высший приоритет коротким задачам; основывается на типе процесса – не зависимо от использованных ресурсов.

– **Динамическое** – приоритет процесса изменяется, как функция разницы между необходимой и фактически полученной услугами. Процесс перемещается по приоритетным группам в зависимости от израсходованного времени и ресурсов.

- **Адаптивно-рефлективное планирование** – предполагается контроль за реальным использованием RAM.

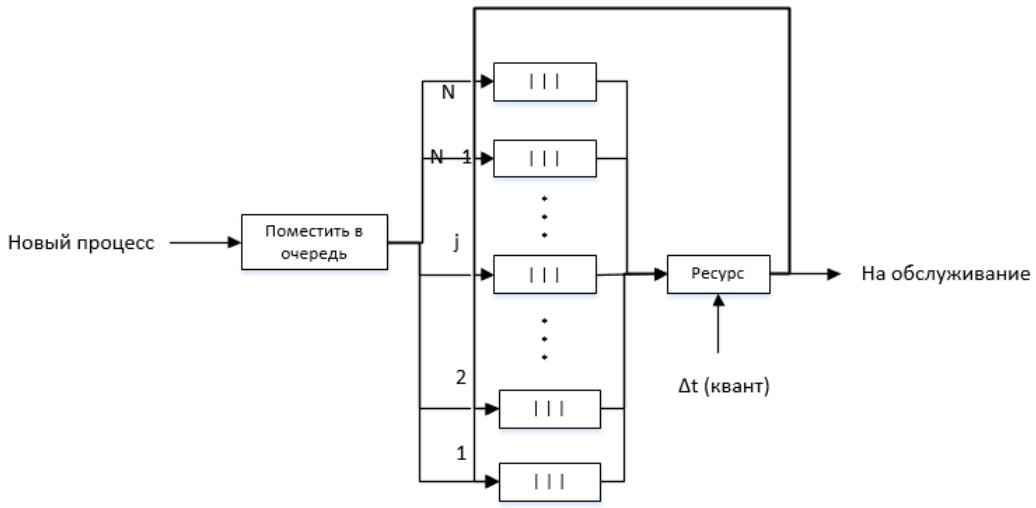

Рис. 15. Пример структуры адаптивно-рефлективного планирования

К началу планирования для каждого процесса устанавливаются ограничения на использование памяти и виртуального времени ЦПУ. ОС отслеживает работу каждого процесса (рабочую область) в течение всего времени его выполнения. Ограничения на память определяются оценкой текущего объема памяти и вектора изменения этого объема. Эта оценка проводится на основе анализа работы процесса в течение текущего кванта времени. Причем, его величина обратно пропорциональна максимальному объему памяти, запрашиваемому процессом. Идея такого подхода ориентирует ОС на процессы с минимальной рабочей областью.

- **Лотерейное (о-ля-ля) планирование** – планировщик случайным образом выбирает процесс, и его обладатель получает доступ к ресурсам. Более важным процессам - большая вероятность выбора планировщиком. Каждый процесс получит процент ресурсов примерно равный проценту его выбора по приоритету. Взаимодействующие процессы могут обмениваться приоритетами для более быстрой обработки очереди планировщиком;
- **Вытесняющее планирование** – при необходимости текущий процесс может быть вытеснен другим процессом. При этом, вытесненный процесс повторно обрабатывается планировщиком. Стратегия с вытеснением может чередоваться с стратегией без вытеснения.

  В данном виде планирования активно используются флаги, отражающие возможности вытеснения себя/другого процесса. Этот алгоритм применим только к процессам с фиксированными приоритетами.

3) **Многоочередные дисциплины обслуживания**

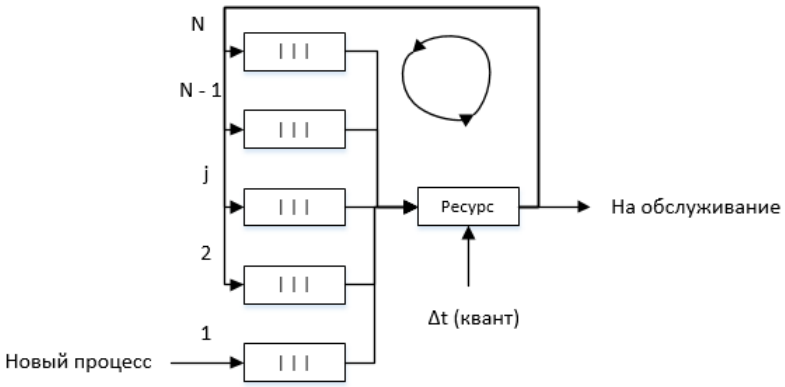

Рис. 16. Пример простой многоочередной дисциплины обслуживания

Основной алгоритм:

а) Образуются N очередей;

б) Все новые процессы поступают в конец первой очереди;

в) Процесс из очереди i поступает на обслуживание тогда, когда все очереди от 1 до $N-1$ пусты;

г) В случае, если процессу из i-очереди не хватило кванта, то он поступает в конец очереди $i+1$;

д) В крайнем случае, он может обслуживаться до конца, а не прерываться, но такое чревато тратой ресурсов и общей задержкой очереди(ей). Особенно, если процесс кривой и написан быдлокодером.

4) **Многоочередная приоритетная дисциплина обслуживания**

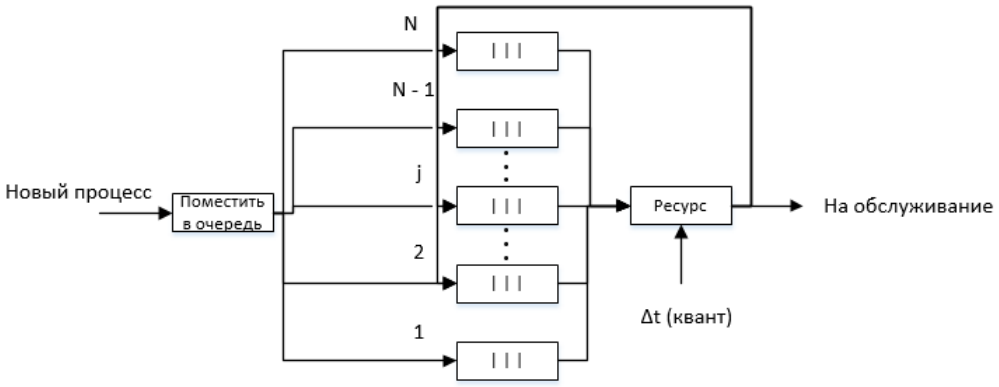

Рис. 17. Пример многоочередной приоритетной дисциплины обслуживания

Все новые процессы поступают в очередь в соответствие с их приоритетом, который хранится в дескрипторе. Во многих ОС, алгоритмы планирования построены с использованием как квантования, так и приоритетов.

В основе планирования лежит квантование, но величина кванта или порядок выбора процесса из очереди зависит от приоритета.

7. СЕДЬМАЯ ЛЕКЦИЯ

7.1. Система очередей планирования ОС NetWare 4.0 фирмы Nowell

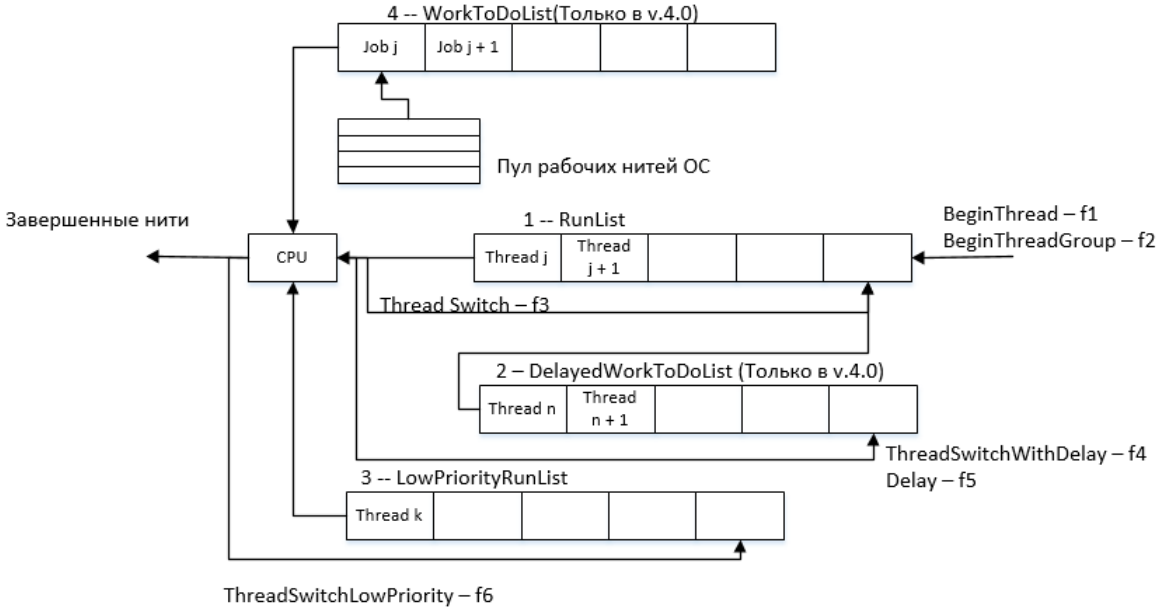

Рис. 18. Схема системы очередей планирования ОС NetWare 4.0

При создании нитей при помощи функций f1, f2 (Begin Thread, Begin Thread Group) – нить попадает в очередь RunList. Эта очередь содержит готовые к выполнению нити [состояние «готовность» на графе существования процесса]. Планировщик выбирает для выполнения стоящую первую в очереди нить и запускает. Нить, завершившая свою очередную итерацию, помещается в конец одной из очередей в зависимости от того, какой вызов передачи управления был использован:

1) В конец очереди 1 – при вызове f3;

2) В конец очереди 2 – при вызове f4, 45;

3) В конец очереди 3 – при вызове f6.

Если нить завершила свою работу, выполнив функцию *return*() в главной функции нити, то она уничтожается.

Нить, находящаяся в очереди 2 после завершения условий ожидания, помещается в конец очереди 1 [состояние «ожидание» на графе существования процесса]. Нити, находящиеся в очереди 3 запускаются только в том случае, если очередь 1 пуста. Обычно, в эту очередь помещаются нити, выполняющие не срочную фоновую работу.

Скринсейвер, асинхронная запись на диск, etc.

Очередь 4 – самая приоритетная. Но она может «забить» другие очереди, поэтому планировщик позволяет выполниться подряд только нескольких нитям из очереди 4, а, затем, выполняет нить из очереди 1. Очередь 4 повышает производительность NLM-приложений (***NLM*** – ***N****owell* ***L****oad* ***M****odule*).

Рассмотренный механизм организации многонитевой работы сочетается с средствами синхронизации.

семафоры, сигналы, etc.

7.2. Синхронизация и взаимодействие процессов

Многие процессы находятся в взаимосвязи с другими процессами.

Печать файлов – печатает все файлы, имена которых другие программы записывают в общедоступный файл заказов. Имеется особая переменная NEXT - также доступная всем процессам и содержит номер первой свободной позиции в файле заказов. В итоге, пострадает процесс S, т.к. R в 4-ю позицию файла заказов запишет свои данные и процесс S не будет выведен. Эту коллизию именуют *«эффектом гонок»*.

Критическая секция – часть программы, в которой осуществляется доступ к разделенным данным.

Взаимное исключение – исключение эффекта гонок. Обеспечивают, чтобы в каждый момент в критической секции находилось не более одного процесса.

Самый простой способ – позволить процессу, находящемся в критической секции, запретить все прерывания. Недостаток - опасно доверять управление системы пользовательскому процессу.

Другой способ – использование блокирующих переменных. С каждый разделяемым ресурсом

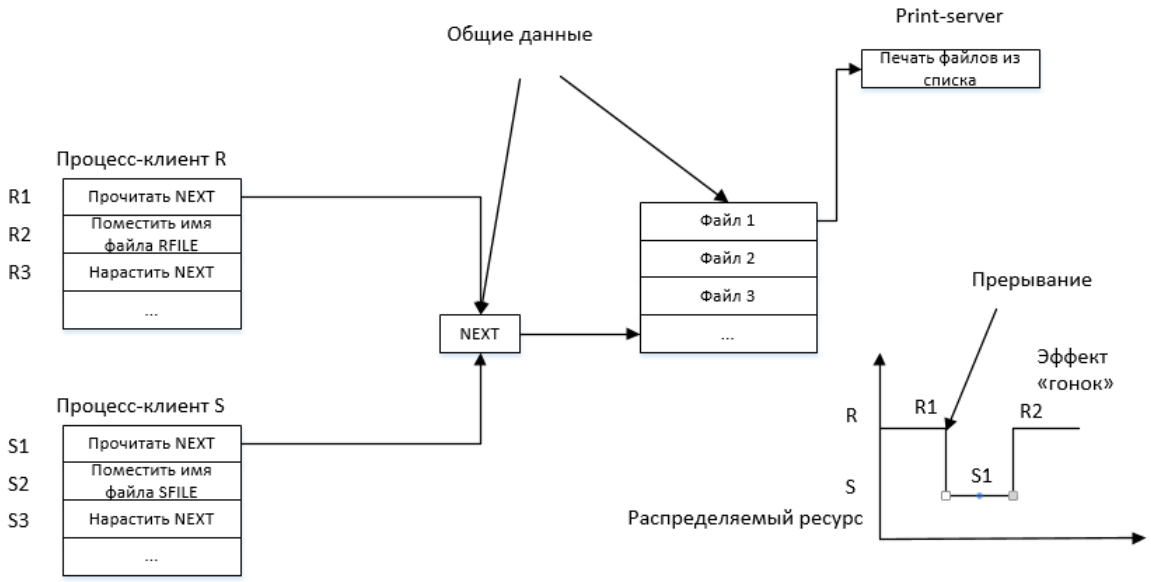

Рис. 19. Пример PrintServer – печати файлов

связывают двоичную переменную. Если 0 – занят, 1 – свободен. Однако, операция проверки-установки блокирующей переменной должны быть неделимой. Иными словами, нужно запрещать все прерывания при выполнении данной операции. Недостаток - бесполезно тратится процессорное время при опросе блокирующей переменной.

Взаимная блокировка (**Дедлоки**/**Клинчи**/**Тупики**) – ситуация, когда конкуренция из-за ресурсов может привести в состояние взаимной блокировки, т.е. все процессы не могут продолжать работу.

Можно выделить 3 метода восстановления восстановления системы:

1) Через откат – в ближайшую точку – в случайном случае;

2) Через принудительную выгрузку ресурсов;

3) Kill -9, в общем - уничтожение процесса, вызвавшего клинч.

Проблемы синхронизации могут быть решены с помощью управлением событиями и памятью.

Синхронизация – сигнализация между процессами по определенному протоколу. Такая операция не зависит от времени и ее принято называть **событием** – т.е. объект синхронизации, используемый для информирования потоков о том, что что-то произошло. В ОС допускается возникновение нескольких событий – каждое событие имеет ID, называемый флагом события – определяющим возможность синхронизации. Число флагов определяется на этапе проектирования.

Если статус < 0, то это значит, что событие прекращено из-за ошибки и системное слово содержит системный код ошибки.
Если статус > 0, то это значит удачное завершение.
Если статус $= 0$, то это событие в данной момент выполняется.

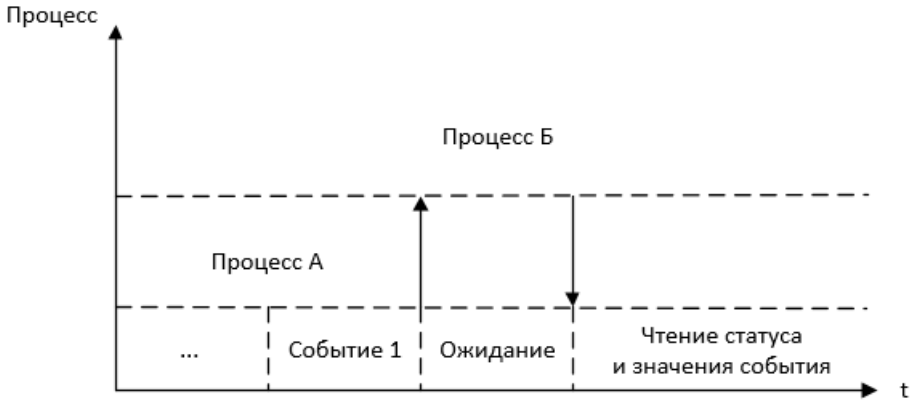

Рис. 20. Взаимодействие процессов

Процесс А – прародитель, Б – подчиненный процесс. А инициализирует Б с помощью события 1, и после завершения, процесс Б посылает статус и значение события 1.

При решении общих задач, процессы должны иметь возможность взаимодействовать – т.е. обмениваться данными. Передаваемую последовательность данных называют сообщением.

Такая связь создает новые проблемы – касается адресации процессов, влияет на структуру планировщика диспетчера, etc. Эта передача информации также усложняется асинхронностью выполнения процессов, т.к. предполагается, что один процесс выполняется со скоростью, независящей от скорости других процессов.

Принимающий сообщение процесс может в данный момент не готов к приему. Для решения – используется временный буфер.

В целом, обмен между процессами может быть разделен на два класса:

1) **Разделяемые переменные – совместно используемые.**

   # События, семафоры

   Семафор – счетчик числа доступных ресурсов (число сообщений в очереди)/неотрицательные целые переменные, используемые для синхронизации.

   Для семафоров определены две операции (примитива) – P и V. Пусть S – семафор, тогда операция V(S) увеличивает S на 1. При этом, прерывания запрещены. P(S) уменьшает S на 1, если это возможно. В противном случае, поток, вызывающий операцию P(S) ждет, когда уменьшение станет возможно. Также тут прерывания запрещены.

   Проверка и уменьшение/увеличение – являются неделимыми (нет прерываниям!).

   Если S может принимать значение только 0 или 1, то семафор превращается в блокирующую переменную (двоичный семафор).

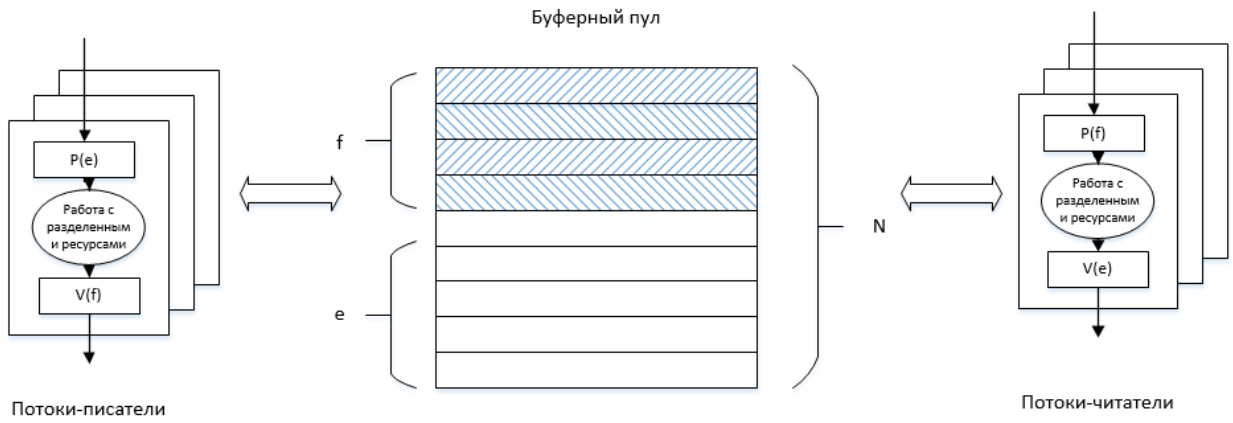

Рис. 21. Взаимодействие буферного пула

Пусть буферный пул имеет размер N, а каждый буфер пула имеет одну запись. Поток-писатель и поток-читатель могут иметь разные скорости и во время работы могут приостанавливаться в зависимости от ситуации. Использование двоичных переменных позволяет обеспечить доступ к критическому ресурсу только одному потоку.

2) **Сообщения.**

Мьютексы похожи на семафоры. Там используются только две операции – захват мьютекса и освобождение.

Мьютекс может быть захвачен не более, чем одним потоком управления, называемый владельцем. Освободить мьютекс может только его владелец, поэтому их не используют в функциях обработки прерываний. Если мьютекс захвачен, то при попытке захвата его другим потоком, этот поток будет приостановлен и помещен в очередь к мьютексу.
В очереди может находиться несколько потоков, которые упорядочены по приоритетам. Если потоков с одним приоритетом несколько, то их упорядочивают по времени установки в очередь.

В основном, мьютексы используют в управлении доступом к ресурсам, которые совместно используются разными потоками. При использовании средств синхронизации может случиться, что менее приоритетные потоки будут мешать выполнению более приоритетных.

Пусть имеется 3 потока – высокого, среднего и низкого приоритета. При этом, выполняется поток имеющий средний приоритет, а высокоприоритетный ждет. Во избежании инверсии приоритетов, приоритет потока управления, владеющий мьютексом, может быть временно повышен.

Имеются 2 подхода при обмене сообщениями:

а) С процессами, имеющими общего прародителя;

б) С любыми.

UNIX имеет также 2 уровня – верхний и нижний. Распределение процессора производится по приоритетной схеме. Осуществляется динамическое изменение приоритетов в зависимости от характеристик процессов на соотношении $\frac{t_{\text{использ.}}}{t_{\text{прознозир.}}}$. Перераспределение времени происходит детерминировано с дискретностью, примерно, 1 раз в секунду. Такой период называют **квантом мультиплексирования процесса**.

Для синхронизации в UNIX используют специальные файлы для передачи сообщений между процессами. Такие файлы могут наследоваться как общие ресурсы.

Процесс в Windows представляет собой один из типов объектов, который связан с другими объектами (поток, секция совместно используемой памяти, порт, файл, маркер доступа, событие, семафор, etc.).

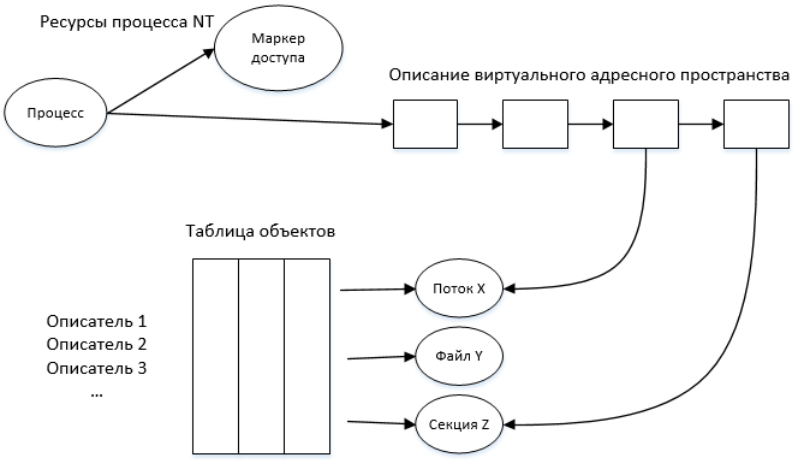

Рис. 22. Процессы в Windows

На высоком уровне абстракции, процесс состоит из исполняемой программы, определяющей начальный код и данные.

Процесс-объект имеет следующие атрибуты и сервисы (или поля и методы):

- **Поля:**
  а) PID;
  б) Маркер доступа;
  в) Базовый приоритет;
  г) Процессорное сродство по-умолчанию;
  д) Размеры квот на ресурсы памяти;
  е) Время выполнения;
  ж) Счетчик ввода-вывода;
  з) Счетчик операций виртуальной памяти;
  и) Порты исключений-отладки;
  к) Коды завершений.
- **Методы:**
  а) Создать;
  б) Открыть;
  в) Установить информацию;
  г) Завершить;
  д) Выделить-освободить виртуальную память;
  е) Защитить виртуальную память;
  ж) Чтение-запись в виртуальную память;
  з) Блокировать-разблокировать виртуальную память.

8. ВОСЬМАЯ ЛЕКЦИЯ

8.1. Управление оперативной памятью

Каждый процесс считает, что его память начинается с нулевого адреса и является непрерывной. Функции, реализующие задачи управления RAM зависят от вида памяти.

Основные функции управления:

1) Отображение или перевод адресов логической памяти на адреса физической памяти – распределение физ.адресов $\overline{0; K}$ среди M процессов;

   Выделяют следующие случаи перевода:

   а) **Абсолютная трансляция** – выполняется компилятором или ассемблером при подготовке программы и генерации ими абсолютных адресов;

   б) **Статическая трансляция** – проводится, если программа составлена в форме выполняемого модуля. Отдельные программы собираются вместе и им присваиваются адреса, установленные относительно некоторого некоторого физического адреса в памяти;

   в) **Динамическая трансляция** – реальные адреса расположения программы в RAM определяются ОС; предполагается, что пространство памяти, отводимое процессу, может меняться в ходе его работы.

2) Расширение границы логического пространства памяти за границы физического – используется overlay-программирование (разные части программы используют одинаковый набор логических адресов). Существуют другие методы, такие как: обмен, свопинг, сцепление, etc. При этих методах логическое адресное пространство каждой программы соответствует физическому пространству (либо на диске, либо в RAM);

3) Разделение.

   # Между программой пользователя и ОС, а также функция распределения (выделение каждому пользовательскому процессу физической памяти);

4) Защита.

   # Защита информации пользователя, ОС, etc.

8.2. Система распределения RAM

Основная идея работы системы:

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

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

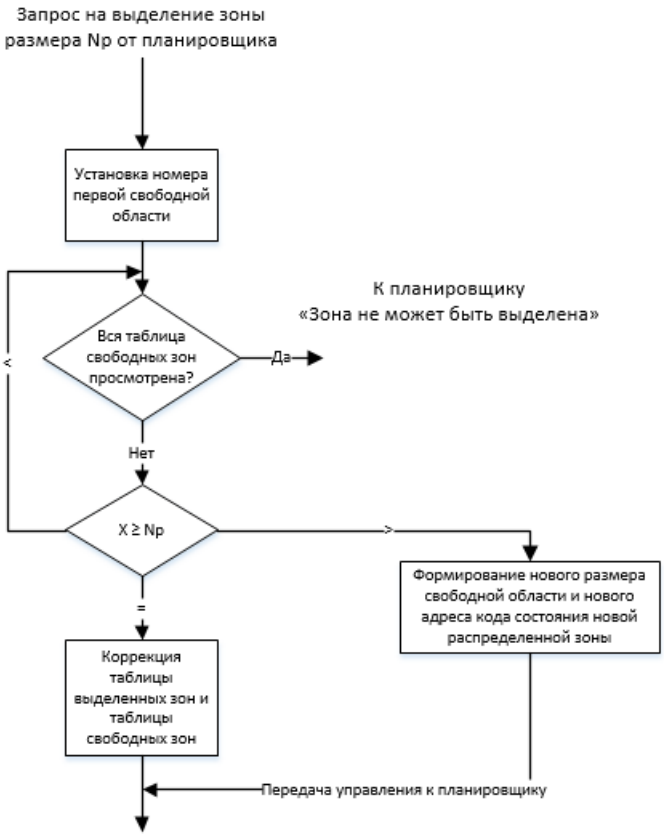

Рис. 23. Система распределения RAM

Если ОС двухуровневая, то память на верхнем распределяется статически, а на нижнем – динамически. На каждом уровне решаются три задачи:

1) Выделение памяти;

2) Учет памяти;

3) Возврат памяти.

Обычно ядро ОС располагают в начальных адресах памяти. Целью алгоритмов распределения является минимизация разделов и числа пустых участков памяти – читай, борьба с фрагментацией. Вне зависимости от алгоритмов, потери из-за фрагментации все равно были, есть и будут.

Алгоритмы основаны на выделении непрерывной и единственной зоны. RAM разбивается на фиксированное число зон до начала работы ОС. Зоны могут быть разными и каждый процесс может выполняться только в пределах своей зоны. Размеры зон могут меняться при работе. Например, зона может отводиться каждый раз своя при долгосрочном планировании, при этом могут формироваться новые зоны из свободных участков. Минимальный размер свободного участка при это ограничен.

Основное условие: нельзя определить зону, размер которой больше RAM.

Задачами ОС при реализации данного метода управления являются:

1) Ведение таблиц свободных и занятых зон, в которых указываются начальные адреса и размеры участков памяти;

2) При поступлении нового процесса должен проводиться анализ запроса, просмотр таблицы и выбор раздела, размер которого удовлетворяет требованиям;

3) Загрузка процесса в выделенный раздел и корректировка таблиц свободных и занятых зон (см. рис. 23).

Алгоритм оптимального размещения – минимизировать объем свободного пространства, остающегося после каждого распределения. Здесь для запроса выделяется свободный участок минимально возможного размера.

$V_z{}^2 = V_z{}^1 - V_x{}^1 \rightarrow V_z{}^3 = V_z{}^2 - V_x{}^2 \rightarrow$ etc.

Здесь:

- V_z – размер свободного участка (остаток);
- Верхний индекс – итерация;
- V_x – требуемый размер участка.

Каждый раз ОС сравнивает остаток с некоторой пороговой величиной и вырабатывает решение – проводить подобной разделение или нет.

9. ДЕВЯТАЯ ЛЕКЦИЯ

9.1. Способы учета свободного пространства

Для управления RAM (в частности, принимать решения) требуется обладать информацией. Поэтому центральное место занимает способ учета свободного пространства.

Один из способов – все пустые участки объединить в список определенного вида. В элементе списка, в начальных и конечных адресах которой описываются характеристики участка и ссылки на другие списки.

Рис. 24. Пример объединения участков в список(?)

Если список упорядочен по размеру – позволяет определить самый подходящий для процесса участок. Всего участки упорядочены в порядке возрастания начальных адресов.

Преимущество – легко обнаружить смежные пустые участки, которые можно объединить в одну.

В случае же двунаправленного – сокращается время поиска.

Поиск на практике чаще всего осуществляется по принципу «первый подходящий свободный участок».

9.2. Свопинг

Используется для повышения эффективности RAM за счет внешней. При нехватке RAM ненужные в данным момент разделы копируются на диск – откачка. Обратный процесс – подкачка. Вообще, грубо – откачка + подкачка = свопинг. Сложность – один раздел может использоваться несколькими процессами.

Наличие свопинга позволяет:

1) Заново распределять память для процесса, не запуская его с самого начала;
2) Появляется возможность выполнять больше малоактивных процессов, чем может одновременно разместиться в RAM;
3) Освободить память, занимаемую процессом, требующего вмешательство пользователя;
4) Более эффективно использовать другие, кроме RAM, ресурсы;

   # В приоритетном планировании.

5) В многопользовательских системах, когда используется один и тот же код программы (не данных), в свопинге могут участвовать только области данных пользователя.

10. ДЕСЯТАЯ ЛЕКЦИЯ

10.1. Overlay

Программа пользователя может быть разбита на главный (корневой) сегмент и один или более неосновных фиксированной длины – **overlay**.

Корневой сегмент *резидентен* – постоянно находится в RAM – а overlay находится на диске и загружаются, по необходимости, в RAM. Область памяти программы пользователя, зарезервированная под overlay, называется **overlay-группой**. Внутри группы overlay фиксированной длины, но разные группы могут иметь разные длины.

По принципу организации и способу загрузки, overlay-системы подразделяются на:

1) **Автоматическая** – деление программы на overlay определяет ОС;
2) **Полуавтоматическая** – пользователь сам определяет, как должна быть разделена программа, а ОС определяет когда и как подгружать overlay;

3) **Программная** – пользователь определяет не только разделение, но и отправляет запросы на подгрузку overlay.

Программа может быть разделена на overlay в виде дерева, где корневой сегмент представляет собой нулевой уровень дерева, а overlay, вызываемый из корневого сегмента принадлежит первому уровню дерева, etc.

10.2. Схема механизма физической адресации

Форма задания адреса при механизме физической адресации полностью определяется механизмом доступа к элементам хранения. Механизм физической адресации – простейший случай доступа, реализованный аппаратно.

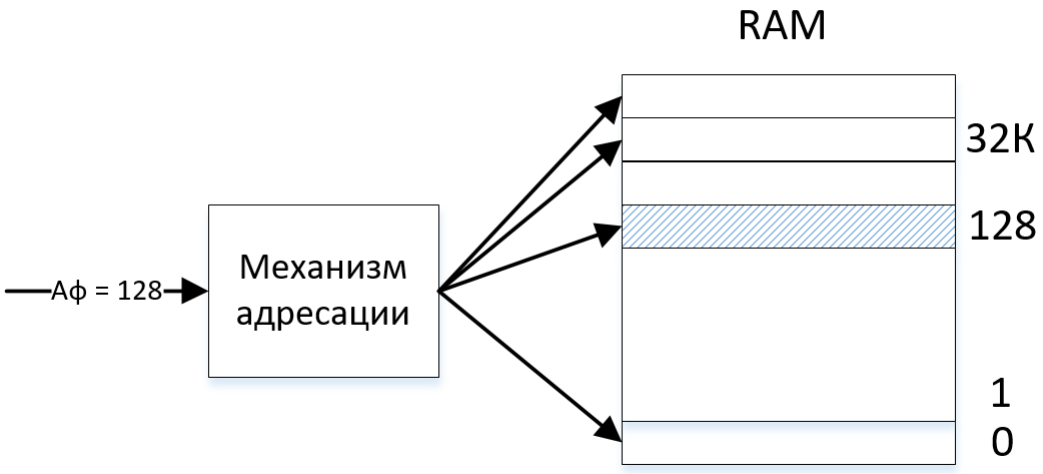

Рис. 25. Механизм физической адресации

Адрес – число, который однозначно определяет номер требуемого элемента хранения. Размер адресного пространства здесь всегда равен объему конкретного вида памяти. Данный механизм служит основой для более сложного механизма доступа, который реализуется, как правило, в программно-аппаратной форме. В этом случае, адрес может задаваться в удобной форме пользователю (или в удобной для управления RAM). Такие адреса называют виртуальными. И пользователи имеют доступ только к программному слою механизма.

11. ОДИННАДЦАТАЯ ЛЕКЦИЯ

11.1. Организация виртуальной RAM

Различают 2 класса схем структуризации адресного пространства:

1) Страничной структуризации;

2) Сегментной структуризации.

11.1.1. Схема структурирования фиксированными страницами

Если размер адресного пространства кратен 2, то и размер страницы. Реально размер страницы – 4 КБ. Это позволяет упростить механизм преобразования адресов.

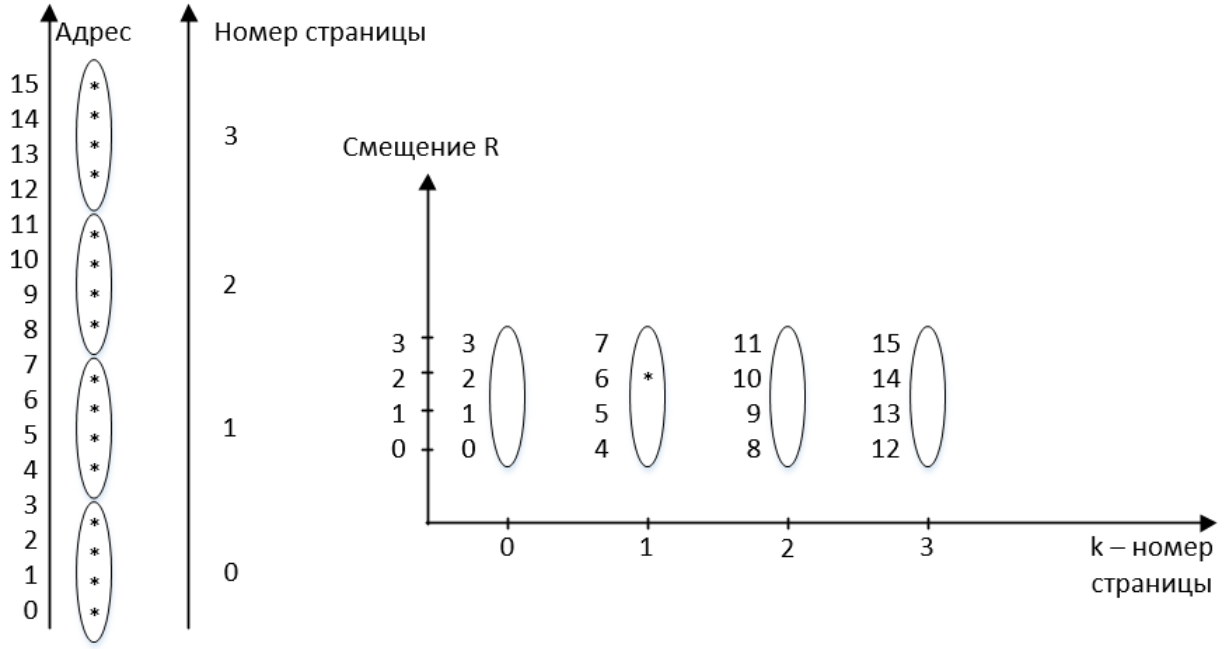

Рис. 26. Структурирование фиксированными страницами

Для перехода из страничного в исходное одномерное адресное пространство используем выражение $A = k \cdot L + R$, где A – размер страницы, k – номер страницы, L – число адресов, входящее в каждую страницу, R – смещение, внутри страницы.

Адрес задается парой величин $(k; R)$.

$\#\ (1,\ 3) \to 7.$

Можно, впрочем, обойтись и одним действием – если воспользоваться двоичным представлением, то пользуемся смещением. Т.е. из двух двоичных чисел при помощи конкатенации получают третье число:

$\#\ 1 \to 0\ 1;\ 3 \to 11. \Rightarrow 0111 = 7$

11.1.2. Механизм работы страничной структуризации

Во время загрузки процесса, часть его виртуальных страниц помещаются в RAM, а остальные в ROM, причем, смежные виртуальные страницы не обязательно располагаются в смежных физических. ОС создает для каждого процесса информационную структуру, называемую **таблицей страниц**. В таблице устанавливаются соответствия между номерами виртуальных и физических страниц (только для страниц в RAM). В противном случае, создается отметка о том, что виртуальная страница выгружена на диск.

В таблице страниц содержится следующая управляющая информация:

1) Признак модификации страницы.

   Если страница не была модифицирована, а лишь подгрузилась из диска и там осталась, то ничего не нужно производить;

2) Признак запрета на выгрузку;

3) Признак обращения к странице;

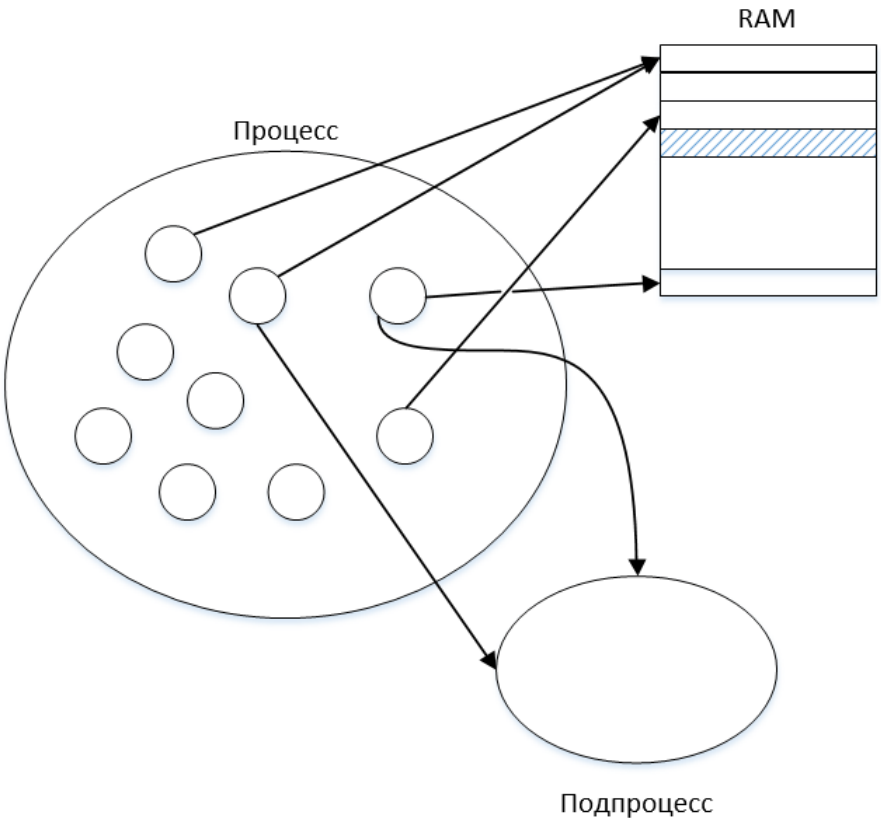

Рис. 27. Страничная структуризация

Время преобразования виртуального адреса в физический определяется временем доступа к таблице страницы.

Уменьшается фрагментация, т.к. будет меньше перераспределений и программа может загружаться в несмежные области. Недостаток – не происходит расширение за границы физического адресного пространства.

Рис. 28. Механизм преобразования

При обращении к RAM выполняются следующие действия:

1) Зная начальный адрес таблица страниц (он хранится в регистре адреса таблицы страниц), номер виртуальной страницы и длину записи в таблице находят требуемую запись;

2) Считывается номер физической страницы;

3) К номеру физической страницы присоединяется смещение при помощи конкатенации.

11.1.3. Схема структурирования переменными страницами

Отсутствует взаимнооднозначные соответствия между структурированным и непрерывным адресами. Здесь по значению непрерывного адреса нельзя однозначно сказать, к какой странице он принадлежит.

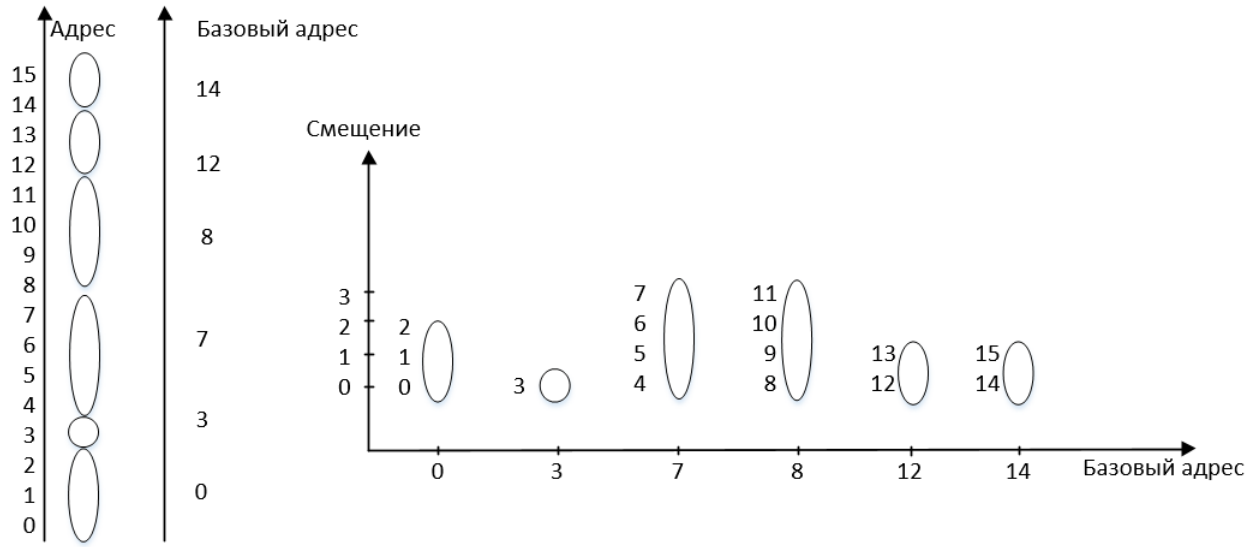

Рис. 29. Структурирование переменными страницами

Для получения значения непрерывного адреса по заданному структурному задается парой $(A_i; R)$ и используется выражение $A = A_i + R$, где A_i – адрес страницы, R – смещение. Т.е. используется принцип «база + смещение».

11.1.4. Схема сегментной структуризации

Сегменты формируются различных размеров, и их номера не упорядочены и представляют собой произвольные целые числа. В представлении добавлено еще и смещение внутри сегмента, но на практике задают сегменты парой (s, R) – номер сегмента и смещение.

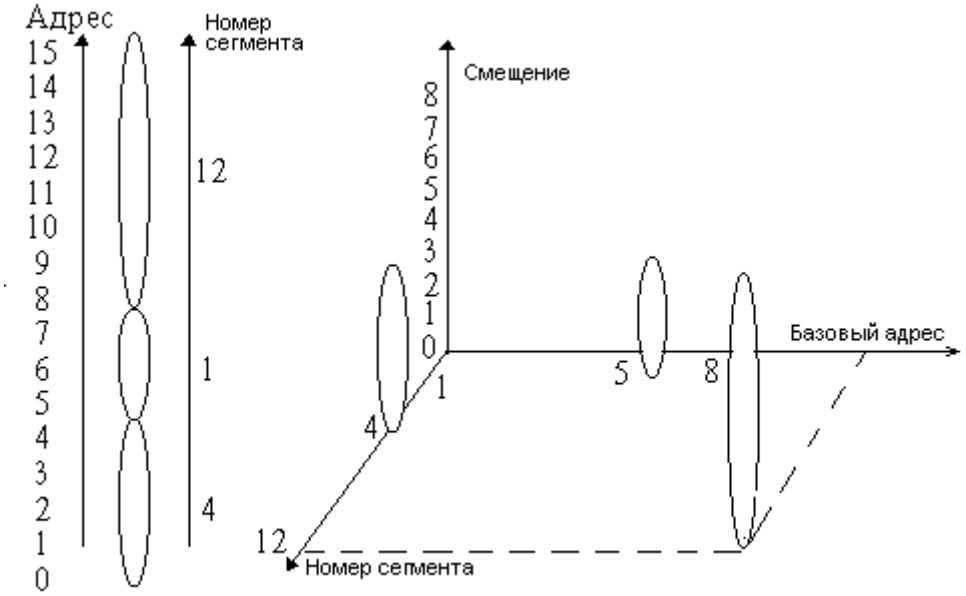

Рис. 30. Сегментная структуризация

В этом случае при переходе к непрерывному адресу необходимо выполнить операции:

1) Сегменту с номером s присваивают базовый адрес A_{s_0};

2) Применяется принцип «база + смещение»: $A + A_{s_0} + R$.

Отдельный сегмент может представлять собой подпрограмму, массив данных, etc. – т.е., что определяется программистом. Иногда, правда, сегментация программы выполняется компилятором по-умолчанию.

При загрузке часть сегментов подгружается в RAM (при этом может использоваться алгоритм оптимального размещения – см. рис. 27.). Оставшаяся часть сегментов подгружается в ROM. При этом сегменты могут занимать несмежные участки. Для учета во время загрузки создается таблица сегментов процесса, где для каждого сегмента размещается следующая управляющая информация:

1) Начальный физический адрес сегмента в RAM;

2) Размер сегмента;

3) Правила доступа к сегменту;

4) Признак модификации сегмента;

5) Признак обращения к сегменту за последний квант времени.

Если виртуальное адресное пространство нескольких процессов включает один и тот же сегмент, то в таблице сегментов этих сегментов делаются ссылки на один и тот же участок RAM. Сегмент же не дублируется и находится в единственном экземпляре.

11.1.5. Схема сегментно-страничной структуризации

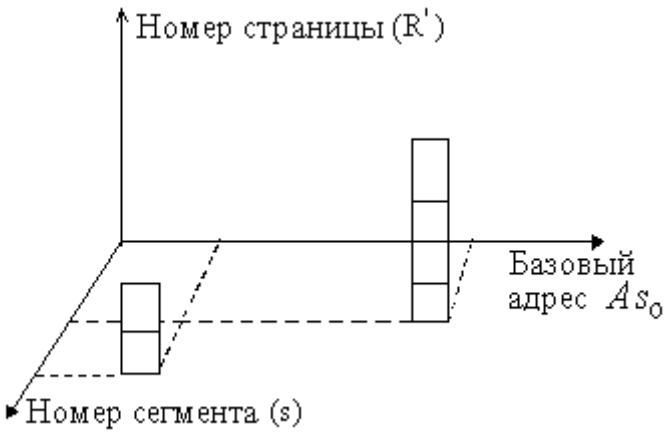

Рис. 31. Сегментно-страничная структуризация

Основы построения сегментно-страничной структуризации:

1) Исходное пространство структурируют фиксированными страницами;

2) Выделяют сегменты, где каждый сегмент рассматривается, как некая непрерывная последовательность номеров страниц. В итоге размер сегмента представляет собой число страниц в нем;

3) Каждому новому сегменту присваивают уникальный ID – s;

4) В пределах каждого сегмента проводится перенумерация страниц по-возрастанию;

5) Сегменту назначается базовый адрес A_{s_0}.

В итоге, адрес указывается с помощью 4-х координат: s – номер сегмента, A_{s_0} – базовый адрес, R' – смещение в сегменте (номер страницы), R – смещение в странице.

Базовый адрес страницы в составе страницы определяется как $A_{R_{\text{базовый}}} = A_{s_0} + R' \cdot L$ (L – размер страницы (длина)).

Если размер страницы кратен 2, то к базовому адресу производят конкатенацию смещения R для получения адреса.

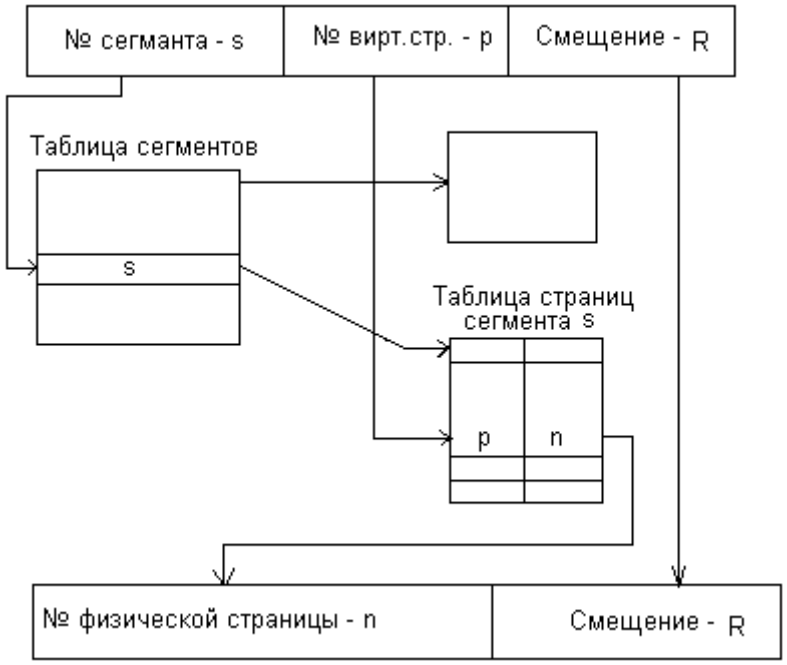

Рис. 32. Механизм преобразования

Для каждого процесса создается своя таблица сегментов. Адрес таблицы загружается в специальный регистр ЦПУ, когда процесс становится активным.

Основные цели использования страничной и сегментной организаций:

1) **Страничная организация**

   Ориентирована, в первую очередь, на удовлетворение нужд системы (нижний уровень). Это позволяет улучшить использование RAM, т.к. уменьшается объем пересылок между рабочей и переменной средами;

2) **Сегментная организация**

   Ориентирована, в первую очередь, на пользователя (верхний уровень) и исполнение сложных многомодульных программ в мультипрограммном режиме.

11.2. Задачи управления виртуальной памяти

Основными задачами управления виртуальной памятью являются:

1) **Размещение.**

   В адресном пространстве RAM выбираются страницы или сегменты, на которые будут отображаться виртуального адресного пространства. Сложность в том, что размер виртуального существенно больше физического пространства;

2) **Перемещение.**

   # Нужно переместить из архивной среды хранения информацию некоторого сегмента в страницу RAM;

3) **Преобразование.**

   # Необходимо найти абсолютный адрес в рабочей среде хранения по его виртуальному в соответствии с функцией преобразования;

4) **Замещение.**

   Связана с необходимостью выбора среди страниц пространства RAM кандидата на перераспределение (см. предыдущий материал).

12. ДВЕНАДЦАТАЯ ЛЕКЦИЯ

12.1. Сегментация с использованием страниц Intel Pentium

Система Intel Pentium поддерживает 16K независимых сегментов. Размер сегмента может быть разный; максимум – 1 млрд. 32-х разрядных слов (word) – 4 ГБ. Для большинства программ важнее размер сегмента, а не количество (статистически) – редко нуждается в более, чем 1000 сегментов.

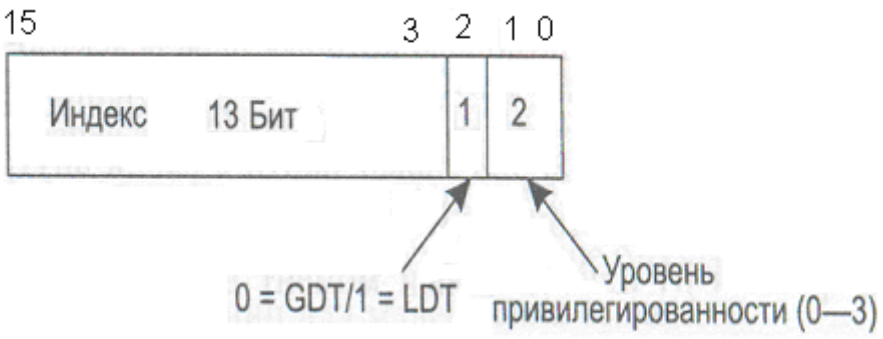

Рис. 33. Селектор в системе Pentium

Основа виртуальной памяти Pentium состоит из двух таблиц:

1) LDT – локальная дескрипторная таблица;

2) GDT – глобальная дескрипторная таблица.

У каждой программы своя локальная таблица дескрипторов, которая описывает локальные сегменты (код, данные, стек, etc.). Глобальная таблица же уникальна – ее совместно используют все программы и она содержит информацию о системных сегментах, включая саму ОС. Для доступа к сегменту, программа сначала загружает селектор сегмента, загружая его в один из 6 сегментных регистров ЭВМ.

Регистр CS содержит селектор для сегмента кода. DS – селектор для сегмента данных.

13 бит (3–15) селектора определяют номер записи в таблице дескрипторов. Каждая таблица ограничена и содержит 8K сегментных дескрипторов. Бит 3 определяет тип сегмента, а 1–2 – уровень привилегированности (отвечают за защиту). Нулевой дескриптор является запрещенным. Если он загружен в сегментный регистр, то это означает, что сегментный регистр не доступен в данный момент. И при попытке использования вызывается прерывание.

Для обеспечения быстрого доступа к элементу таблицы дескрипторов, его сохраняют в микропрограммных регистрах. Дескриптор состоит из 8 байт, в который входит базовый адрес сегмента, его размер, etc.

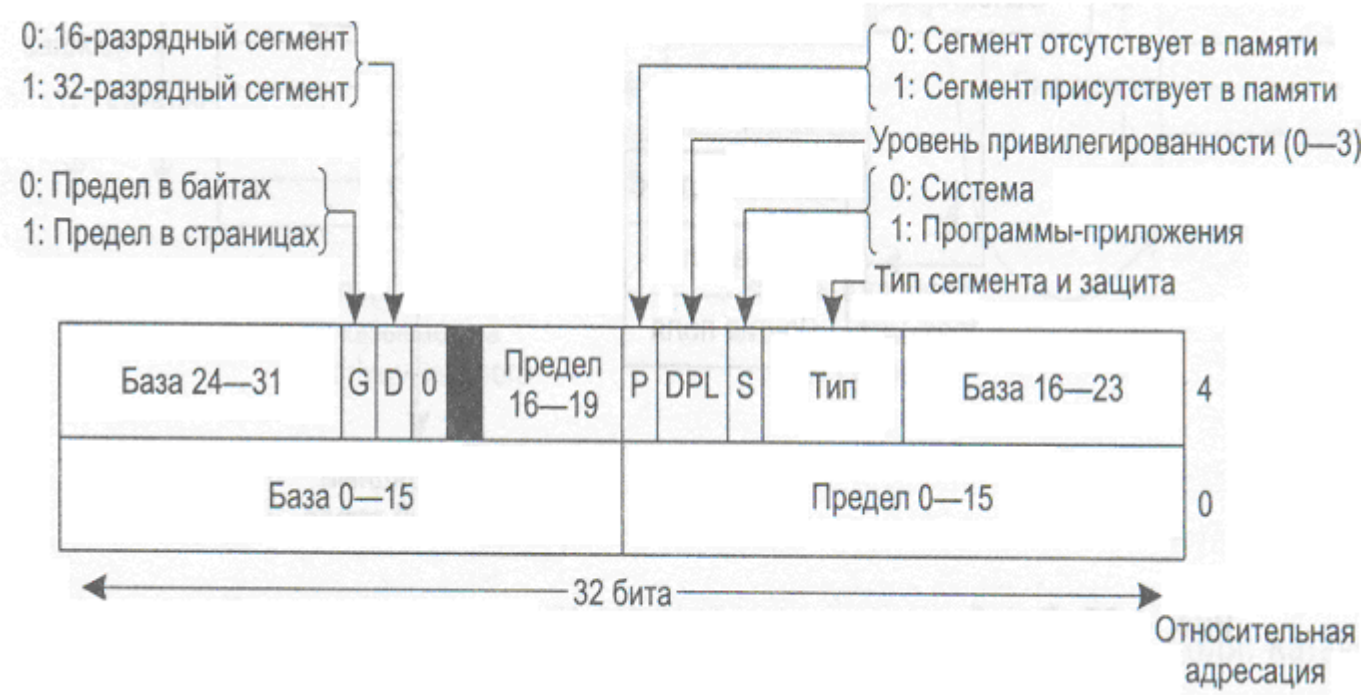

Рис. 34. Дескриптор программного сегмента в системе Pentium

Сначала выбирается LDT (GDT) на основе второго бита селектора. Затем, селектор копируется во внутренний рабочий регистр и три младших бита сбрасываются в «0». Далее, к селектору прибавляется адрес одной из таблиц, чтобы получить прямой указатель на дескриптор. Пара «селектор–смещение» преобразуется в физический адрес следующим образом:

1) Как только программа узнает, какой сегментный регистр используется, она находит в своих внутренних регистрах находит полный дескриптор, соответствующий этому сегменту. Если сегмент не существует (в этом случае селектор – «0») или выгружен, то возникает прерывание;

2) Программа прерывает, выходит ли смещение за пределы сегмента. Если да – прерывание;

3) В дескрипторе есть 32-х разрядное поле, определяющее размер сегмента, но доступно лишь 20 бит – т.к. их достаточно для сегментов размером 2^{32};

4) Если сегмент находится в памяти, а смещение попало в нужный интервал, то Pentuim прибавляет поле «база» в дескрипторе к смещению, и, в итоге, формируется линейный адрес.

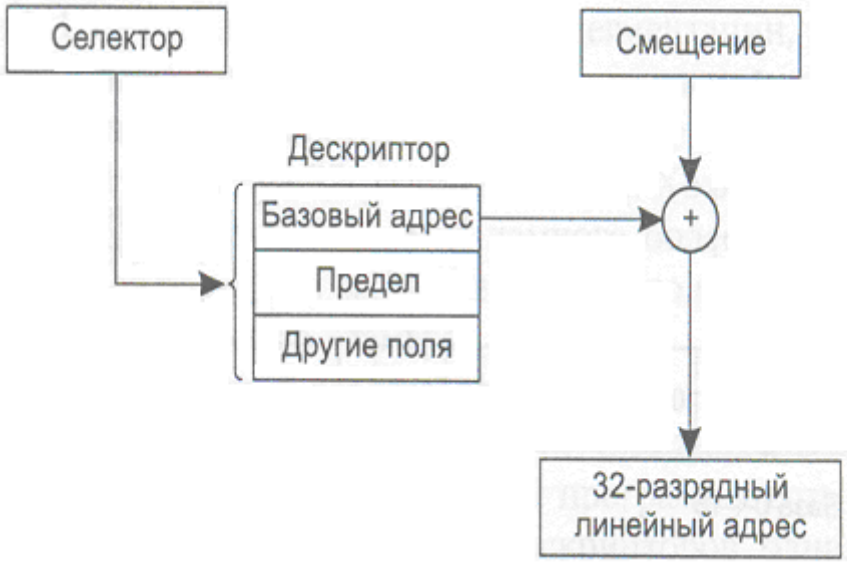

Рис. 35. Преобразование пары «селектор-смещение» в физический адрес

Поле «база» в дескрипторе разбито на 3 части – это сделано для совместимости с процессорами Intel 80286, где это поле имело всего 24 бита. Если разбиение на страницы блокировано (с помощью битов в глобальном управляющем регистре), то линейный адрес интерпретируется, как физический адрес и посылается в память для чтения или записи. Т.е. при отключенной страничной схеме памяти, получаем чистую схему сегментации с базовым адресом каждого сегмента, который находится в его дескрипторе. Сегментам разрешено перекрываться случайным образом. А если доступна страничная подкачка, то линейный адрес интерпретируется как виртуальный – и отображается на физический адрес с помощью таблиц страниц.

Трудность состоит в том, что при 32-х разрядном виртуальном адресе и странице размером 4 КБ, сегмент может содержать 1 млн. страниц. Для ускорения поиска, используется двухуровневое отображение с целью уменьшения числа страниц для малых сегментов. У каждой работающей программы есть страничный каталог, состоящий из 1024-х 32-х разрядных записей.

12.2. Организация виртуальной памяти в Windows

ОС Windows построена в соответствие с классическими принципами (сегментно-страничная структуризация). В основе лежит страничная организация.

Для платформы Alpha – 8 КБ.

В общем виде, схема описывается следующим образом:

Линейный адрес разбивается на несколько частей – старшая содержит номер элемента в корневой таблиц. Этот элемент содержит, в свою очередь, адрес таблицы следующего уровня. Следующая часть линейного адреса содержит номер элемента в этой таблице – etc. до последней таблицы, содержащей номер физической страницы. А самая младшая – номер байта в физической странице.

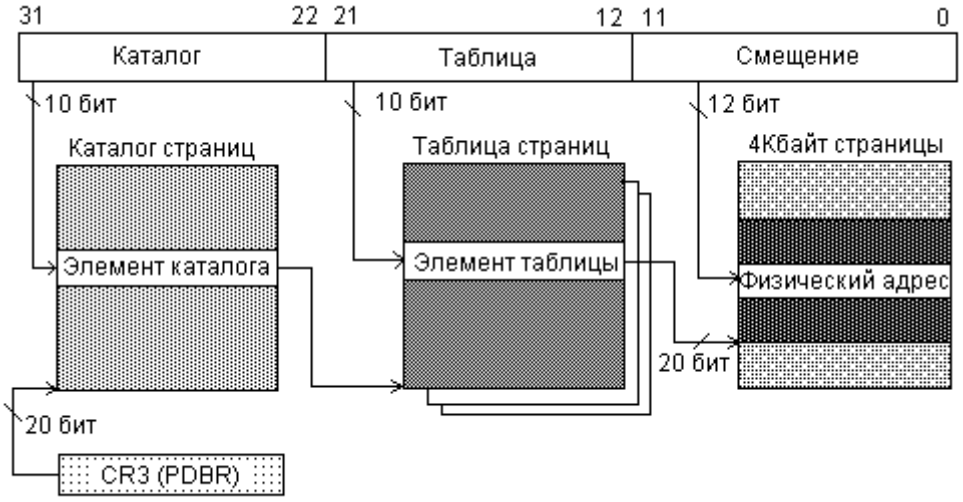

Рис. 36. Размещение данных в сегментно-страничной структуризации Windows

Старшие 10 бит определяют один из 1024-х элементов в каталоге страниц.

Рассмотрим отдельный элемент таблицы страниц:

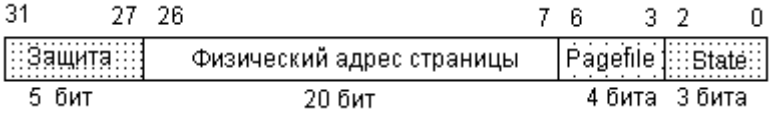

Рис. 37. Элемент таблицы страниц

Старшие 5 бил определяют тип страницы с точки зрения допустимых операций.

Win32 API Поддерживает 3 допустимых значения этого поля: PAGE_NOACCESS, PAGE_READONLY, PAGE_READWRITE

С битов 3-6 описывается файл подкачки.

Комбинация этих битов определяет один из 16 возможных в системе файлов. Младшие биты определяют состояние страницы в системе. Старший бит T определяет страницу как переходную. Следующий бит D определяет страницу, в которой была произведена запись. Младший бит P определяет присутствие страницы в RAM или Swap. Для ускорения страничного преобразования, с ЦПУ есть специальная кэш-память **TLB** (***T**ranslation **L**ookaside **B**uffer*), где хранятся наиболее часто используемые элементы каталога и элементы таблиц страниц.

Каждый процесс Windows имеет свой каталог страниц и адресное пространство. Это дает дополнительные возможности по защите процессов.

12.3. Файловые системы

Файл – поименованная целостная совокупность данных на внешнем носителе, на которую наложена некоторая структура.

Файловая система – часть ОС, обеспечивающая выполнение операций над файлами. Одной из традиционных задач файловой системы является сокрытие реального местоположение информации от пользователей на физическом уровне. Другой задачей является обеспечение независимости программ от конкретной конфигурации ЭВМ.

Функции ФС:

1) Реализация метода(ов) организации данных;
2) Перевод логических характеристик в физические;
3) Защита пользователя

   # от случайных сбоев оборудования;
4) Обеспечение возможности одного накопителя между несколькими пользователями;
5) Защита от несанкционированного доступа;
6) Шифрование;
7) Etc.

13. ТРИНАДЦАТАЯ ЛЕКЦИЯ

13.1. Упрощенное взаимодействие в файловой системе

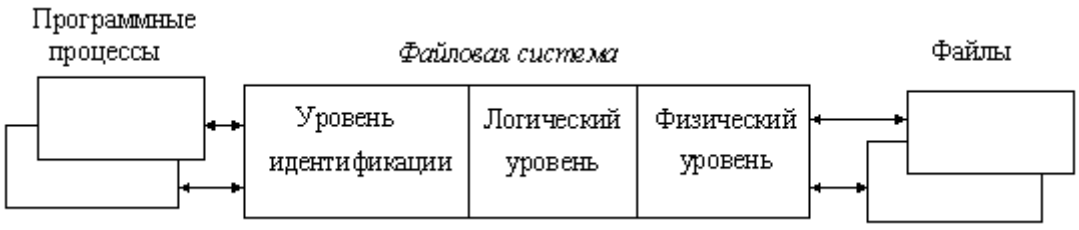

Рис. 38. Взаимодействие в файловой системе

На уровне идентификации выполняются следующие действия:

1) По символьному имени файла определяется его уникальное имя (в ОС, где один файл может иметь несколько имен);
2) По уникальному имени определяются атрибуты файла;
3) Сравниваются полномочия пользователя/процесса с правами доступа к файлу.

На логическом уровне выполняются следующие действия:

1) Определяются координаты логической записи в файле;
2) Алгоритм работы зависит от конкретной модели.

На физическом уровне определяется номер физического блока, который содержит требуемую логическую запись.

Метод доступа к файлу – способ адресации его составным элементом на основе логической структуры файла.

Характеристики файлов:

1) Имя файла – используются разные форматы.

   # Формат имени 8.3 (8 имя, 3 расширение). Длина пути – 80 символов. В современных – длинные имена (255 имя, 260 символов длина пути – Windows);

2) Расширение ОС должна распознавать стандартный набор расширений;

3) Атрибуты файлов. Специфицируют тип файла, защиту и способ буферизации.

   # Пароль доступа, владелец, создатель, признак только для чтения, системный, архивный, скрытый, etc. , длина записи, текущий размер;

4) Тип файла:

   а) Файл может быть сегментированный (обеспечивает произвольный доступ и может иметь неограниченный размер);

   б) Непрерывный файл – обеспечивает один непрерывный блок – доступ к содержимому за одно обращение. Рост неограниченный, минимальный учетный уровень служебной информации.

   Файлы бывают:

   а) Обычные файлы (текстовые, двоичные);

   б) Файлы-каталоги (справочники), содержащие списки файлов и их характеристики;

   в) Специальные файлы (байт-ориентированные, блок-ориентированные).

      # В GNU/Linux – символьные устройства, блочные устройства, именованные каналы для взаимодействия между процессами, сокет, символьные ссылки, etc.

5) Типы доступа к файлу – на чтение, обновление, запись, удаление, изменение атрибутов, etc.

13.2. Логическая организация файловой системы

Можно выделить две основных иерархических модели:

1) Дерево.

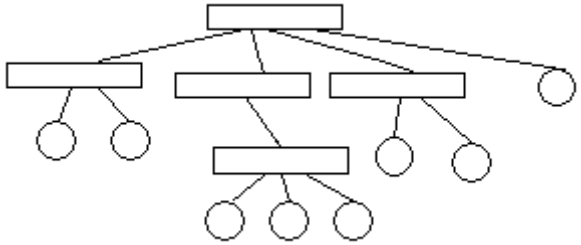

Рис. 39. Иерархическая модель «дерево»

2) Сеть.

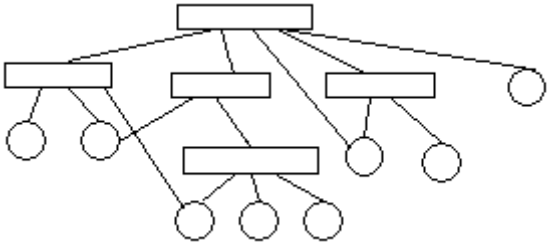

Рис. 40. Иерархическая модель «сеть»

Работа производится на уровне логических записей. Логическая запись – наименьший элемент данных, доступный пользователю для выполнения различного рода операций.

Файл рассматривается, как одномерный массив составных элементов, называемых записями. Длина логической записи может быть как постоянной, так и переменной. Всякая логическая запись имеет свой порядковый номер в составе файла. Доступ к элементам фала производится последовательный. Для организации подобного вида доступа достаточно иметь указатель на последнюю запись.

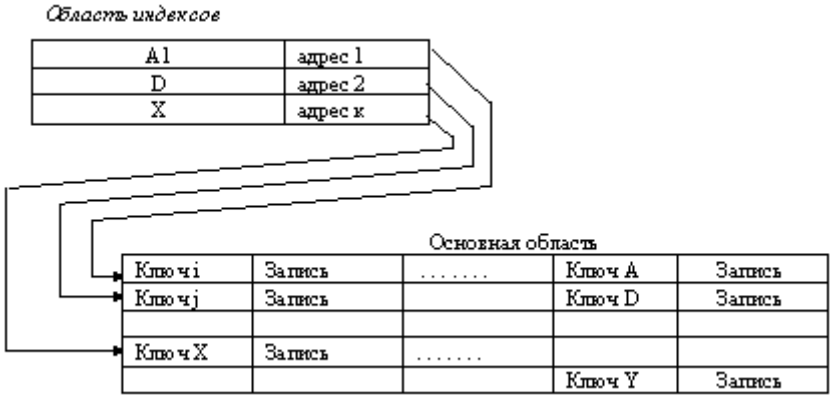

Рис. 41. Индексно-последовательная структура файла

Существует ряд методов, основанных на идентификации файла по некоторому ключу – индивидуальному признаку. Структура файла усложняется, но сокращается число обращений к диску. Кроме данных, вводится дополнительная учетная информация. В данной структуре поиск файлов производится сначала в прямом, а после – в последовательном порядке. Все записи упорядочиваются по значению ключей, выделяют группы записей ключей, расположенных в файле подряд и могут храниться в пределах одной дорожки на диске.

Для более быстрого поиска таких групп, используют специальную структуру, именуемую индексом, при этом каждый элемент индекса описывает одну группу записей. Индекс может содержать значение максимального ключа в группе и ссылку на начальную запись. По индексу находят начало записи требуемой группе, а в ней последовательным анализом ключей находят нужную запись. Вводится область переполнения, куда вводится запись, а из файла только ссылки.

Индекс дорожки.

13.3. Библиотечные структуры файлов

Также имеется два уровня – учетный и информационный. Файл составляется из совокупности последовательных наборов данных, где каждый набор имеет собственное имя в составе данного файла. Такие наборы называют разделами, расположение которых не упорядочено. Разделы записываются в порядке поступления. Расположение раздела фиксируется в каталоге учетного уровня файлов. Сами элементы каталога располагаются в алфавитном порядке. Расположение информации на конкретном носителе чаще всего значительно отличается от логической упорядоченности. Преобразование из логическую в физической структуру осуществляется на основе информации, сосредоточенной в каталогах и на основе специальных описателей – дескрипторов.

Алгоритмы работы с оперативной памятью и с внешней похожи между собой. Разница – в реализации (с учетом специфики). Например, организация свободных участков в упорядоченный список неудобна по причине большого числа обращений к диску. В связи с этим, указатели на элементы списка как правило объединяют в таблицу и загружают ее в память для дальнейшего использования. Внешняя память разбивается на блоки фиксированного размера. Каждый блок имеет свой уникальный порядковый номер. Блок – наименьшая единица данных, участвующая в процессе обмена между устройством внешней памяти и RAM.

14. ЧЕТЫРНАДЦАТАЯ ЛЕКЦИЯ

14.1. Способы размещения структуры

1) Непрерывное размещение.

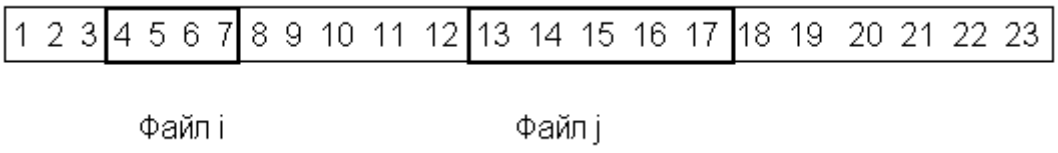

Рис. 42. Непрерывное размещение структуры

Файл состоит из последовательности блоков диска, которые образуют единый сплошной участок.

Достоинства – адрес файла определяется номером начального блока, простота реализации и быстрый доступа.

Недостаток – заранее неизвестна длина файла.

2) Связный список блоков.

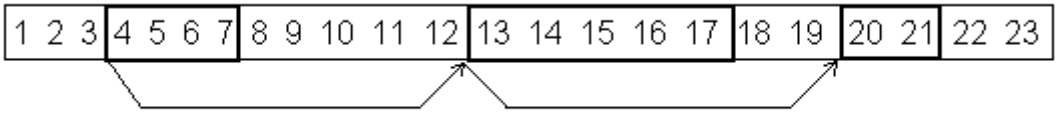

Рис. 43. Структура в виде связного списка блоков

В начале каждого блока содержится указатель на следующий блок файлов.

Достоинства – адрес файла определятся номером начального блока, отсутствует практически фрагментация, файл может изменяться.

Недостаток – сложность реализации доступа к произвольной части файла, больше памяти на служебную информацию

3) Связный список индексов

С каждым блоком связывается индекс. Индексы располагаются в отдельной области диска – например, таблица размещений файлов **FAT** (***F**ile **A**llocation **T**able*). Это позволяет отслеживать состояние различных участков дискового пространства.

После выхода Windows 95 появилась 32-х битная **VFAT** (***Virtual FAT***) – усовершенствованная FAT, в которой разрешены длинные имена. После – FAT32. Она отличается от VFAT только количественными параметрами (меньше кластеры, больший размер диска, не ограничений на число файлов в корневом каталоге, etc.).

Здесь сохраняются достоинства предыдущего способа, но при том исчезает недостаток – т.е. чтобы обеспечить доступ к произвольной позиции, достаточно прочитать блок индексов.

Недостаток – больше памяти на служебную информацию.

4) Перечень номеров блоков.

   Номера блоков просто перечисляются.

   Достоинства – отсутствует практически фрагментация, снимаются проблемы динамического расширения.

   Недостаток – усложняется алгоритм распределения и поиска.

В UNIX реализован вариант, позволяющий обеспечить фиксированную длину адреса не зависимо от размера файла.

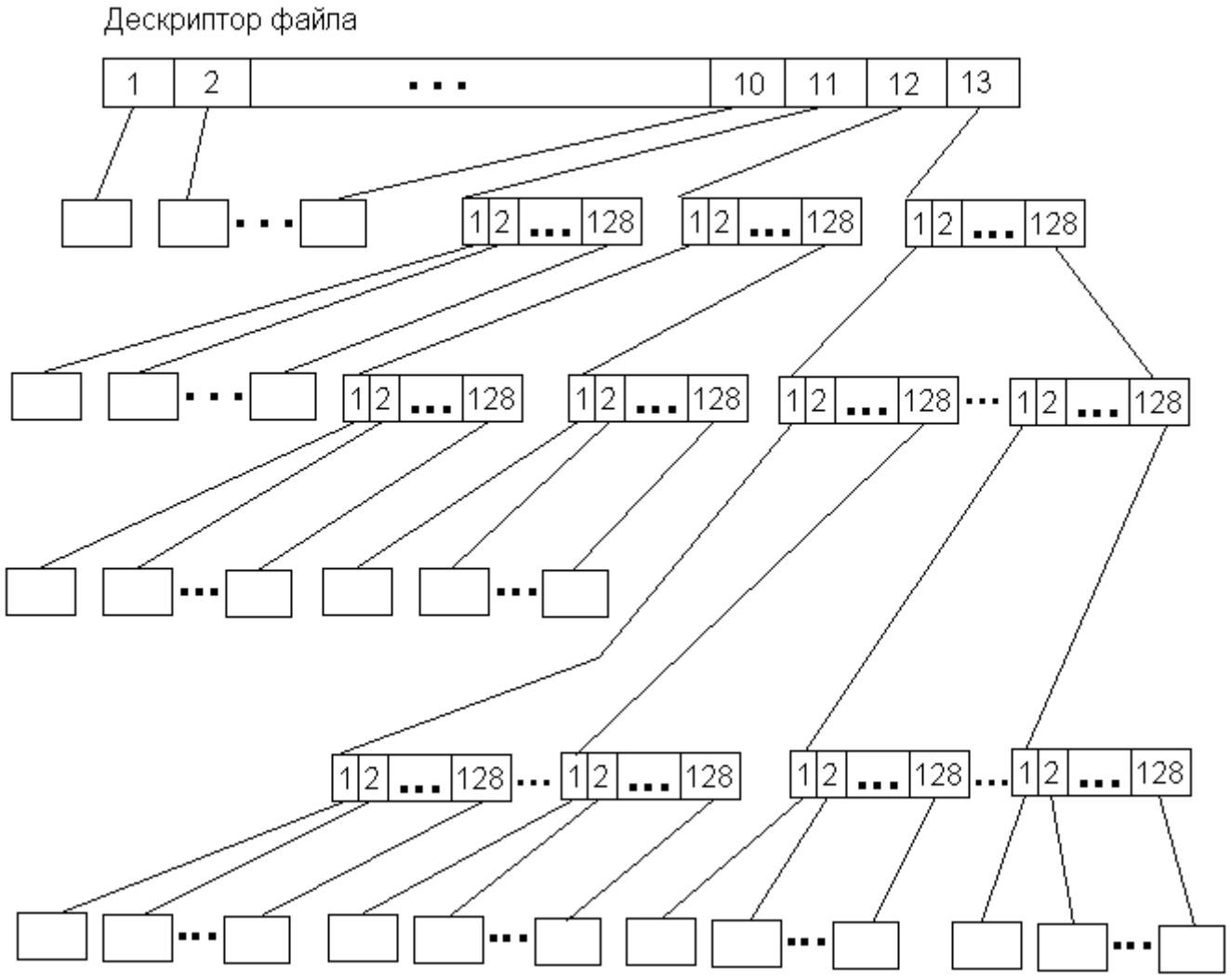

Рис. 44. Структура в UNIX

Каждый файл в системе имеет дескриптор, в составе которого хранится список, содержащий 13 номеров блоков диска. Используется как прямая, так и косвенная адресация. Первые 10 элементов списка непосредственной указывают на 10 блоков файла. Если недостаточно – используют оставшиеся элементы списка. 11 – для одноуровневой косвенной адресации. В нем указан номер блока, хранящий 128 номеров блоков, которые могут принадлежать файлу. Это чилсо блоков возможных принадлежать файлу. Блок – 512Б/1КБ/2КБ/4КБ.

14.2. Права доступа к файлу

Определить права доступа – определить для пользователя доступные операции над файлами. В разных ОС определен свой список операций доступа.

Можно выделить следующие операции:

1) Создание

2) Уничтожение

3) Открытие

4) Закрытие

5) Запись

6) Установка атрибутов файла

	Файл 1	Файл 2	.....	Файл N
Пользователь 1	Читать	Выполнять Читать		Создать
Пользователь 2	Читать Писать	Выполнять		Уничтожать

Рис. 45. Права доступа к файлу

Пользователи могут быть разделены на категории. В GNU/Linux – владельцы, члены группы, остальные. В целом выделяют два основных подхода по отношению к правам доступа– избирательный (для каждого пользователя и файла сам владелец определяет допустимые операции), мандатный (системы устанавливает права пользователя по отношению к каждому разделяемому ресурсу в зависимости от группы пользователя).

14.3. Кэширование диска

При работе с внешними устройствами используется подсистема буферизации, работающей по принципу кэш-памяти. Запрос к внешнему устройству, в котором адресация осуществляется блоками, может быть перехвачен подсистемой буферизации

Такая система представляет собой **буферный пул** – совокупность однородных динамически распределяемых блоков RAM одинаковой длины – и комплекс программ, управляющих пулом. Каждый буфер пула имеет размер равный одному блоку.

Механизм работы: при поступлении запроса на чтение некоторого блока, подсистема буферизации просматривает сперва свой буферный пул. В случае обнаружения нового блока, производится копирование оного в буфер запрашивающего процесса (т.е. без обращения к ROM). В противном случае – считывание и передача процессу (одновременно) с записью в буфер.

При отсутствии свободного буфера, на диск вытесняется самая редко используемая информация.

Windows – размер кэша может меняться в зависимости от ситуации. Кэш увеличивается при интенсивной работе в сети или уменьшается при снижении активности в сети и запуске большого числа приложений.

15. ПЯТНАДЦАТАЯ ЛЕКЦИЯ

15.1. Структура современной файловой системы

Современная ОС может работать с несколькими файловыми системами, имеет многоуровневую структуру и работает следующим образом: приложение обращается только через переключатель (Windows – устанавливаемый диспетчер ФС – IFS). Переключатель преобразует запросы в формат следующего уровня конкретной файловой системы. Каждый драйвер ФС (ДФС) поддерживает определенную организацию файловой системы. ДФС позволяет сразу нескольким приложениям выполнять операции над файлами.

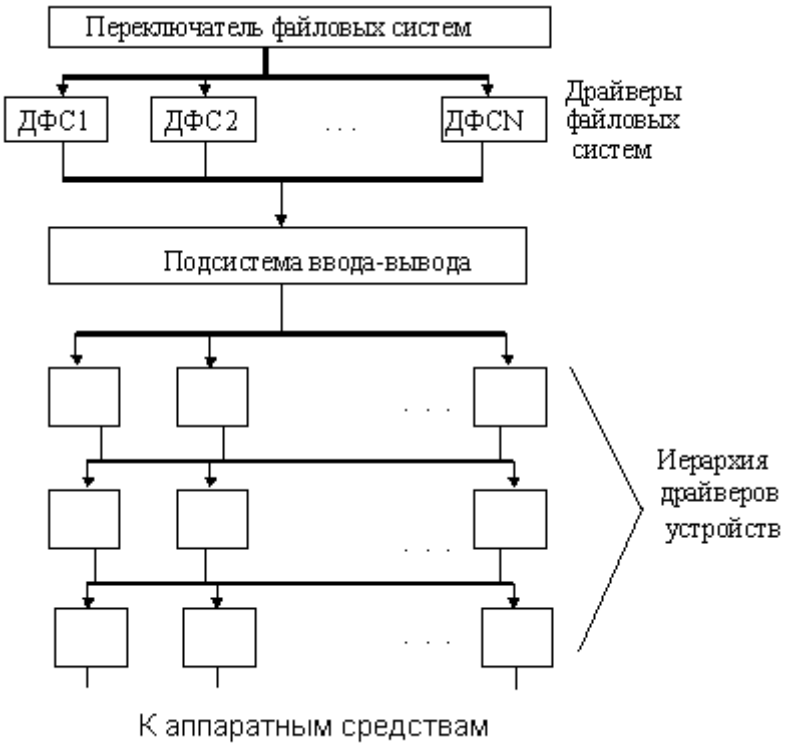

Рис. 46. Структура современной файловой системы

Подсистема ввода-вывода отвечает за загрузку, инициализацию и управление всеми модулями нижних уровней (драйверы портов). Данная подсистема постоянно присутствует в памяти и обеспечивает совместную работу иерархии драйверов устройств. Каждый уровень драйверов устройств представляет собой определенный тип:

1) Драйвер жестких дисков;

2) Драйверы, перехватывающие запросы к блочным устройствам;

3) Драйверы портов, управляющие конкретными адаптерами.

15.2. Примеры файловых систем

Windows работает с FAT12, FAT16, FAT32, exFAT, NTFS4, NTFS5, ReFS, CDFS. И дополнительно имеет распределенную ФС DFS – расширение сетевого сервиса и позволяет объединить в единый логический том сетевые ресурсы, расположенные в разделах с различными ФС. EFS представляет собой надстройку над NTFS.

15.2.1. FAT16

Для простых структуры, небольших дисков размером до 2 ГБ. Таблица размещается в начале тома. В нем, в целях защиты, хранится 2 копии таблиц. Таблица и корневой каталог находятся по строго фиксированным адресам. Таблица используется для определения кластеров файла. Похоже на оглавление. Каталог – специальные файл с 32-х битными элементами для каждого файла, содержащимся в нем.

Элемент для каждого файла включает:

1) Имя файла (8 + 3);
2) Байт атрибута (1 – идентификация, как подкаталог, 2 – метка тома, остальные – указывают на какой-либо вариант использования);
3) Время модификации (16);
4) Дата модификации;
5) Первый размещаемый блок;
6) Размер файла.

Эту информацию используют любые ОС, поддерживающие FAT.

15.2.2. FAT32

Предусматривает ряд специальных областей на диске, получаемых в процессе форматирования.

1) Головная запись загрузки;
2) Таблица разбиения диска;
3) Запись загрузки;
4) Таблица размещения файлов;
5) Корневой каталог.

На физическом уровне пространство диска разбивают на области по 512 байт – секторы. Файлы состоят из кластеров, которые состоят из целого числа секторов, кратных степени 2.

Размер кластера можно рассчитать, поделив объем диска на 64 КБ. Каждый кластер содержит номер следующего кластера, связываясь по цепочке. В итоге, FAT представляет собой БД, связывающую кластеры дискового пространства с файлами. В свою очередь, каталог представляет тоже БД. Доступ реализуется следующим образом: после приема запроса на чтение файла, файловая система сначала просматривает запись каталога для этого файла. Цель – получить первый кластер файла. Затем система обращается к элементу FAT для данного кластера, чтобы найти следующий в цепочке кластер, пока не будет найдет последний кластер.

Объем диска	Размер кластера FAT16 (Кб)	Размер кластера FAT32 (Кб)
256 – 511 Мб	8	4
512 – 1023 Мб	16	4
1024 – 2 Гб	32	4
2 – 8 Гб	-	4
8 – 16 Гб	-	8
16 – 32 Гб	-	16
32 и более	-	32

Рис. 47. Сравнение размеров кластеров FAT16 и FAT32

Размер раздела (Мб)	Количество секторов на кластер	Размер кластера
До 512	1	512 байт
512 – 1024	2	1 Кб
1025 – 2048	4	2 Кб
2049 – 4096	8	4 Кб
4097 – 8192 и т.д.	16	8 Кб
Более 32678	128	64 Кб

Рис. 48. Сравнение числа секторов на кластер с их размером

В FAT32 как элементы FAT, так и имена секторов 32-х разрядные. Это означает, что максимально возможная емкость диска – 2 ТБ.

Из таблиц – разные объемы, в FAT32 более эффективное использование дискового пространства, надежность и скорость загрузки программ.

15.2.3. NTFS

Имеет более эффективные алгоритмы для работы с большими дисками, отказоустойчивость (самовосстановление NTFS). Каждый файл имеет более богатый набор свойств, позволяет назначать права доступа к отдельным файлам. Используются кластеры, размер которых зависит от размера раздела.

Форматирование раздела приводит к созданию нескольких системных файлов и главной таблицы файлов – MFT. MFT содержит информацию о всех файлах и папках в разделе. NTFS – объектно-ориентированная ФС, которая обрабатывает все файлы, как объекты с атрибутами. Частью файла является также информация по описанию самой ФС.

Основные отличия NTFS от FAT:

1) Система безопасности позволяет устанавливать различные права доступа к файлам и папкам для пользователей и групп;

2) Быстрое восстановление тома в случае сбоя;

3) Гибкие опции форматирование – позволяют использовать более эффективно пространство;

4) Тома могут расширяться;

5) Имеются зеркальные тома (Windows NT Server).

NTFS5 имеет дополнительные преимущества:

1) Защита отдельных файлов при помощи шифрования;

2) Расширение томов без перезагрузки;

3) Имеются возможности по отслеживанию распределенных ссылок – сохранение ярлыков при перемещении файлов с одного тома на другой или другую ЭВМ;

4) Квотирование диска – квота на дисковое пространство, доступные на работу каждому пользователю.

В предыдущих версиях любой пользователь имел все пространство дисков в сервере. Квотирование выполняется по каждому тому, поэтому не имеет значения – находятся ли тома на одном физическом диске или на разных устройствах. На NTFS5 имеется оснастка MMC, которая работает с томами, форматированными как в FAT16, FAT32, NTFS, etc.

С помощью оснастки возможно создавать и управлять следующими динамическими томами:

1) Простой том – пространство на одном диске;

2) Составной том – связное пространство на нескольких дисках;

3) Чередующийся том – имеет несколько областей, каждая из которых расположена на отдельном диске, но существует ряд особенностей и отличий: при записи информация «расщепляется» и пишется параллельно на каждый из дисков тома;

4) Зеркальный том – а-ля RAID 1, т.е. отказоустойчивая система – полная копия одного из томов, которая обеспечивает избыточность данных за счет создания этой копии.

Многие идеи NTFS проявились из HPFS, что была введена впервые в OS/2.

Первые 16 секторов раздела составляют загрузочный блок с меткой диска и прочей информации для загрузки системы. В 16-ом располагается «супер-блок», содержащий информацию в целом о файловой системе, а именно – раздел раздела, указатель на корневой каталог, номер версии файловой системы, счетчик элементов каталогов, указатели на список испорченных на диске блоков, таблицы дефектный и доступных секторов. Сектор 17 (SpareBlock) содержит указатель на список секторов, которые можно использовать, счетчик доступных секторов, которые можно использовать для горячего исправления ошибок, резерв свободных блоков для управления деревьями каталогов, языковые наборы символов. Оставшееся пространство раздела делится на полосы, размером по 8 МБ.

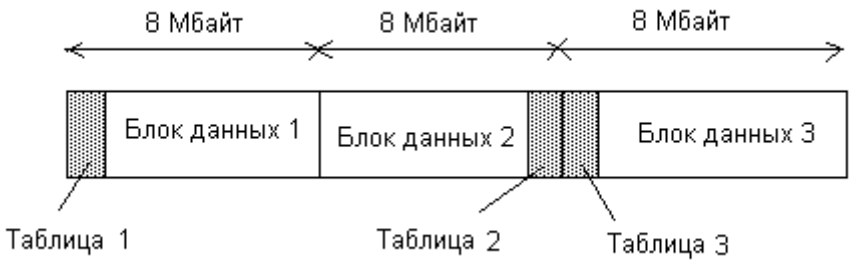

Рис. 49. Пример разделения пространства

Подобный способ увеличивает непрерывное пространство для размещения файла, а в таблицах хранится информация о занятых и свободных кластерах.

16. ШЕСТНАДЦАТАЯ ЛЕКЦИЯ

16.1. RAID

RAID (от англ. ***R**eduntant **A**rray of **I**ndependent **D**isks – избыточный массив независимых дисков*) – система хранения данных, в которой объединяются несколько логических дисков в один логический том для повышения производительности, отказоустойчивости или и того, и другого.

Уровни RAID:

1) RAID 0 – обычный с чередованием и без отказоустойчивости;
2) RAID 1 – зеркальный дисковый массив;
3) RAID 2 – зарезервирован для массивов, в которых применяется код Хэмминга с распределенной четностью;
4) RAID 3 и RAID 4 – дисковые массивы с чередованием и выделенным диском четности (ныне не применяются);
5) RAID 5 – с чередованием и не выделенным диском четности;
6) RAID 6 – дисковый массив с чередованием и двойной контрольной суммой, вычисляемой двумя разными способами;
7) Остальное – подвиды и комбинации предыдущих.

   # RAID 10 – массив RAID 0, построенный из массивов RAID 1; RAID 60 – тоже самое, только из массивов RAID 5.

16.2. Особенности HPFS

Особенности расположения каталогов на диске – резервируется полоса, расположенная в середине диска. Это позволяет добиться, что магнитные головки никогда не проходят более половины ширины диска. Каждый элемент-каталог, описывается специальной структурой – файловым дескриптором, занимающим один сектор и содержащим следующую информацию:

1) Указатель на начало файла;

2) Первые 15 символов имени файла;

3) Время последней записи и последнего доступа;

4) Журнал, содержащий информацию о предыдущих обращениях к файлу;

5) Структура распределения секторов – размещение файла на диске;

6) Первые 300 байт расширенных атрибутов файла (максимум 64).

При открытии файла, в кэш считываются первые 4 сектора автоматически – файловый дескриптор и 3 первых сектора файла.

16.3. Свойства распределенных файловых систем

В основе – клиент-серверная архитектура построения. Под клиентом понимается некоторая ЭВМ, которая обращается к некоторому файлу, а под сервером – ЭВМ, хранящая файлы и обеспечивающая к ним доступ.

Распределенные файловые системы имеют следующие свойства:

1) Сетевая прозрачность – клиенты должны иметь возможность обращаться к удаленным файлам, пользуясь теми же самыми операциями, что и для доступа к локальным;

2) Прозрачность размещения – имя файла не должно определять его местоположение в сети;

3) Независимость размещения – имя файла не должно изменяться при изменении его физического местоположения;

4) Мобильность пользователя – пользователи должны иметь возможность обращаться к разделенным файлам из любого узла сети;

5) Устойчивость к сбоям – система должна продолжать функционирование при неисправности отдельного компонента – сервера или сегмента сети;

6) Масштабируемость – вертикальное или горизонтальное;

7) Мобильность файлов – перемещение файла из одного места в другое в пределах системы.

В распределенных системах решается вопрос по пространствам имен или нескольким состояниям – сохранении одного или нескольких. Вопрос, связанный с методами удаленного доступа – DFS.