Основные понятия дисциплины «Операционные системы»

Появление операционных систем и их функции

История появления и развития системного обеспечения началась с того момента, когда люди осознали, что любая программа требует операций ввода-вывода данных. Это произошло в 50-е годы прошлого столетия. Собственно, операционные системы появились чуть позже. Основной причиной их появления было желание автоматизировать процесс подготовки вычислительного комплекса к выполнению программы.

Для автоматизации труда программиста стали разрабатывать специальные алгоритмические языки высокого уровня, а для автоматизации труда оператора вычислительного комплекса была разработана специальная управляющая программа, загрузив которую в память один раз оператор мог ее далее использовать неоднократно и более не обращаться к процедуре программирования ЭВМ через пульт оператора. Именно эту управляющую программу и стали называть операционной системой. Со временем на нее стали возлагать все больше задач, она стала расти в объеме.

Разработчики стремились к тому, чтобы операционная система как можно более эффективно распределяла вычислительные ресурсы компьютера, ведь в 60-е годы операционные системы уже позволяли организовать параллельное выполнение нескольких программ. Помимо задач распределения ресурсов появились задачи обеспечения надежности вычислений. К началу 70-х годов диалоговый режим работы с компьютером стал преобладающим, и у операционных систем стремительно начали развиваться интерфейсные возможности.

На сегодняшний день операционная система (ОС) представляет собой комплекс системных управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между аппаратурой компьютера и пользователем с его задачами, а с другой стороны, предназначены для наиболее эффективного расходования ресурсов вычислительной системы и организации надежных вычислений.

Основные функции операционных систем:

1. Прием от пользователя (или от оператора системы) заданий, или команд, сформированных на соответствующем языке, и их обработка.
2. Загрузка в оперативную память подлежащих исполнению программ.
3. Распределение памяти, а в большинстве современных систем и организация виртуальной памяти.
4. Запуск программы.
5. Идентификация всех программ и данных.

6. Прием и исполнение различных запросов от выполняющихся приложений.

7. Обслуживание всех операций ввода-вывода.

8. Обеспечение работы систем управлений файлами и/или систем управления базами данных.

9. Обеспечение режима мультипрограммирования, то есть организация параллельного выполнения двух или более программ на одном процессоре, создающая видимость их одновременного исполнения.

10. Планирование и диспетчеризация задач.

11. Организация механизмов обмена сообщениями и данными между выполняющимися программами.

12. Обеспечение взаимодействия связанных между собой компьютеров (для сетевых ОС).

13. Защита одной программы от влияния другой, обеспечение сохранности данных, защита самой операционной системы от исполняющихся на компьютере приложений.

14. Аутентификация и авторизация пользователей. Аутентификация – процедура проверки имени пользователя и его пароля на соответствие тем значениям, которые хранятся в его учетной записи. Авторизация – в соответствии с учетной записью пользователя, который прошел аутентификацию, ему назначаются определенные права.

15. Удовлетворение жестким ограничениям на время ответа в режиме реального времени (характерно для операционных систем реального времени).

16. Обеспечение работы систем программирования, с помощью которых пользователи готовят свои программы.

17. Предоставление услуг на случай частичного сбоя системы.

Операционная система изолирует аппаратное обеспечение компьютера от прикладных программ пользователей. И пользователь, и его программы взаимодействуй с компьютером через интерфейсы операционной системы.

Понятие операционных сред и оболочек

Операционная система выполняет функции управления вычислениями в компьютере, распределяет ресурсы вычислительной системы между различными процессами, и образует ту программную среду, в которой выполняются прикладные программы пользователей. Такая среда называется операционной.

Набор функций и сервисов операционной системы, а также правила обращения к ним как раз и образуют то базовое понятие, которое мы называем операционной средой. Таким образом, термин «операционная среда» означает, прежде всего, соответствующие интерфейсы, необходимые программам и

пользователям для обращения к управляющей (супервизорной) части операционной системы с целью получить определенные сервисы.

Каждая операционная система имеет множество системных функций. Совокупность системных вызовов и правил, по которым их следует использовать, определяет интерфейс прикладного программирования (*API – Application Program Interface*). Очевидно, что программа, созданная для работы в некоторой операционной системе, скорее всего не будет работать в другой операционной системе, поскольку *API* у этих операционных систем, как правило, различаются. Поэтому разработчики операционных систем стали создавать так называемые программные среды.

Программную (системную) среду следует понимать, как некоторое системное окружение, позволяющее выполнить все системные запросы от прикладной программы.

Помимо основной операционной среды в операционной системе организованы (путем эмуляции иной операционной среды) дополнительные программные среды.

Параллельное существование терминов «операционная система» и «операционная среда» вызвано тем, что операционная система (в общем случае) может поддерживать несколько операционных сред.

Операционная среда может включать несколько интерфейсов (оболочек): пользовательские и программные.

Программы-оболочки относятся к классу системных программ. Они обеспечивают более удобный и наглядный способ общения с компьютером, чем штатные средства ОС.

Некоторые программы не заменяют «штатную оболочку», а дополняют ее или добавляют в нее новые функции.

Прерывания

Прерывание – это принудительная передача управления от выполняемой программы к системе (а через нее – к соответствующей программе обработки прерывания), происходящая при возникновении определенного события.

Идея прерывания была предложена в середине 50-х годов. Основная цель введения прерываний – реализация асинхронного режима функционирования и распараллеливание работы отдельных устройств вычислительного комплекса.

Механизм прерываний реализуется аппаратно-программными средствами. Прерывание непременно влечет за собой изменение порядка выполнения команд процессором.

Механизм обработки прерываний подразумевает выполнение шагов (рис.1):

1. Установление факта прерывания.
2. Запоминание состояния прерванного процесса вычислений.
3. Управление аппаратно передается на подпрограмму обработки прерывания.

4. Сохранение информации о прерванной программе, которую не удалось спасти помощью аппаратуры.

5. Собственно, выполнение программы, связанной с обработкой прерывания.

6. Восстановление информации, относящейся к прерванному процессу.

7. Возврат на прерванную программу.

Шаги 1-3 реализуются аппаратно, шаги 4-7 – программно.

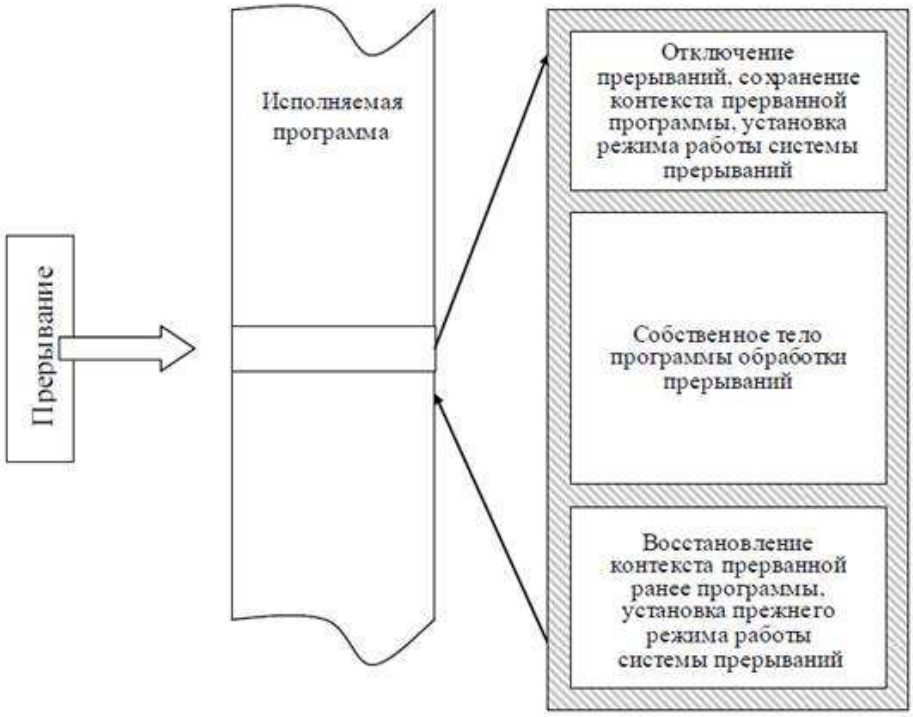


Рис. 1. Механизм обработки прерываний

Прерывания, возникающие при работе вычислительной системы, можно разделить на два основных класса: внешние (их иногда называют асинхронными) и внутренние (синхронные).

Внешние прерывания вызываются асинхронными событиями, которые происходят вне прерываемого процесса.

Внутренние прерывания вызываются событиями, которые связаны с работой процессора, и являются синхронными с его операциями.

Наконец, существуют собственно программные прерывания.

Процессор может обладать средствами защиты от прерываний: отключение системы прерываний, маскирование (запрет) отдельных сигналов прерывания.

Программное управление специальными регистрами маски (маскирование сигнала прерывания) позволяет реализовать различные дисциплины обслуживания:

• с относительными приоритетами, то есть обслуживание не прерывается даже при наличии запросов с более высокими приоритетами.

• с абсолютными приоритетами, то есть всегда обслуживается прерывание с наивысшим приоритетом.

• по принципу стека (последним пришел, первым обслужен).

Основные принципы построения ОС

Среди множества принципов построения операционных систем перечислим несколько наиболее важных:

• *Принцип модульности.*

Операционная система строится из множества программных модулей. Под модулем в общем случае понимают функционально законченный элемент системы, выполненный в соответствии с принятыми межмодульными интерфейсами. По своему определению модуль предполагает легкий способ его замены другим при наличии заданных интерфейсов.

• *Принцип виртуализации.*

Виртуализация ресурсов позволяет не только организовать разделение тех ресурсов между вычислительными процессами, которые не должны разделяться. Виртуализация позволяет абстрагироваться от конкретных ресурсов, максимально обобщить их свойства и работать с некоторой абстракцией, вобравшей в себя наиболее значимые особенности.

• *Принципы мобильности (переносимости) и совместимости.*

Мобильность, или переносимость, означает возможность и легкость переноса ОС на другую аппаратную платформу. Мобильная ОС обычно разрабатывается с помощью специального языка высокого уровня, предназначенного для создания системного программного обеспечения.

Одним из аспектов совместимости является способность ОС выполнять программы, написанные для других систем или для более ранних версий данной ОС, а также для другой аппаратной платформы.

• *Принцип генерации операционной системы из программных компонентов.*

Согласно принципу генерируемости исходное представление центральной системной управляющей части ОС (ее ядра и основных компонентов, которые должны постоянно находиться в оперативной памяти) должно обеспечивать возможность настройки, исходя из конкретной конфигурации конкретного вычислительного комплекса и круга решаемых задач. Под генерацией операционной системы понимается ее сборка (компоновка) из отдельных программных модулей.

В результате генерации получают скомпонованные двоичные коды ОС и построенные системные таблицы, отражающие конкретную конфигурацию компьютера. Эта процедура проводится редко перед достаточно протяженным периодом эксплуатации операционной системы.

• *Принцип открытости*.

Открытая операционная система доступна для анализа как пользователям, так и системным специалистам, обслуживающим вычислительную систему. Наращиваемая (модифицируемая, развиваемая) операционная система позволяет не только использовать возможности генерации, но и вводить в ее состав новые модули, совершенствовать существующие и т. д. К открытым операционным системам прежде всего следует отнести *UNIX*-системы и, естественно, системы *Linux*.

Принцип обеспечения безопасности вычислений

Обеспечение безопасности при выполнении вычислений является желаемым свойством для любой многопользовательской системы. Правила безопасности определяют такие свойства, как защита ресурсов одного пользователя от других и установление квот по ресурсам для предотвращения захвата одним пользователем всех системных ресурсов (таких как память).

Задачей ОС является управление процессами и ресурсами компьютера или, точнее, организация рационального использования ресурсов в интересах наиболее эффективного выполнения процессов. Для решения этой задачи операционная система должна располагать информацией о текущем состоянии каждого процесса и ресурса. Универсальный подход к предоставлению такой информации заключается в создании и поддержке таблиц с информацией по каждому объекту управления.

Общее представление об этом можно получить из рис. 2, на котором показаны таблицы, поддерживаемые операционной системой: для памяти, устройств ввода-вывода, файлов (программ и данных) и процессов. Хотя детали таких таблиц в разных ОС могут отличаться, по сути, все они поддерживают информацию по этим четырем категориям. Располагающий одними и теми же аппаратными ресурсами, но управляемый различными ОС, компьютер может работать с разной степенью эффективности. Наибольшие сложности в управлении ресурсами компьютера возникают в мультипрограммных ОС.

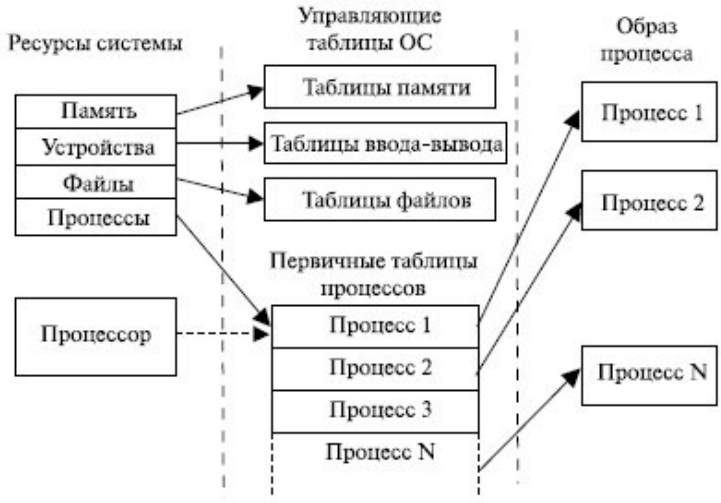


Рис. 2. Таблицы ОС

Мультипрограммирование (многозадачность, *multitasking*) – это такой способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ. Чтобы поддерживать мультипрограммирование, ОС должна определить для себя внутренние единицы работы, между которыми будут разделяться процессор и другие ресурсы компьютера. В ОС пакетной обработки, распространенных в компьютерах второго и сначала и третьего поколения, такой единицей работы было задание. В настоящее время в большинстве операционных систем определены два типа единиц работы: более крупная единица – процесс, или задача, и менее крупная – *поток*, или *нить*. Причем процесс выполняется в форме одного или нескольких потоков.

Вместе с тем, в некоторых современных ОС вновь вернулись к такой единице работы, как *задание* (*Job*), например, в *Windows*. Задание в *Windows* представляет собой набор из одного или нескольких процессов, управляемых как единое целое.

Процессы рассматриваются ОС как заявки или контейнеры для всех видов ресурсов, кроме одного – процессорного времени. Это важнейший ресурс распределяется операционной системой между другими единицами работы – потоками, которые и получили свое название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд. Каждый процесс начинается с одного потока, но новые потоки могут создаваться (порождаться) процессом динамически. Когда поток завершает работу, он может прекратить свое существование. Процесс завершается, когда прекратит существование последний активный поток.

Возникает вопрос: зачем нужна такая сложная организация работ, выполняемых операционной системой? Ответ нужно искать в развитии теории и практики мультипрограммирования, цель которой – в обеспечении максимально эффективного использования главного ресурса вычислительной системы – центрального процессора (нескольких центральных процессоров).

Мультипрограммирование призвано повысить эффективность использования вычислительной системы. Однако эффективность может пониматься по-разному. Наиболее характерными показателями эффективности вычислительных систем являются:

• *пропускная способность* – количество задач, выполняемых системой в единицу времени;

• *удобство работы пользователей*, заключающихся, в частности, в том, что они могут одновременно работать в интерактивном режиме с несколькими приложениями на одной машине;

• *реактивность системы* – способность выдерживать заранее заданные (возможно, очень короткие) интервалы времени между запуском программы и получением конечного результата.

В зависимости от выбора одного из этих показателей эффективности ОС делятся на системы пакетной обработки, системы разделения времени и системы реального времени (некоторые ОС могут поддерживать одновременно несколько режимов).

Системы *пакетной обработки* предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Максимальная пропускная способность компьютера достигается в этом случае минимизацией простоев его устройств и прежде всего процессора.

В *системах разделения времени* пользователям (в частном случае – одному) предоставляется возможность интерактивной работы сразу с несколькими приложениями. Для этого каждое приложение должно регулярно получать возможность "общения" с пользователем. Эта проблема решается за счет того, что ОС принудительно периодически приостанавливает приложения, не дожидаясь, когда они "добровольно" освободят процессор. Всем приложениям попеременно выделяются кванты времени процессора, таким образом, пользователи, запустившие программы на выполнение, получают возможность поддерживать с ними диалог со своего терминала. Если время кванта выбрано достаточно небольшим, то у всех пользователей складывается впечатление единоличной работы на машине.

Системы *реального времени* предназначены для управления техническими объектами (спутник, ракета, атомные электростанции, станок, научная установка и др.), технологическими процессами (гальваническая линия, доменный процесс и т.п.), системами обслуживания разного рода (резервирование авиабилетов, оплата покупок и счетов и др.). Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа управления объектом. В противном случае возможны нежелательные последствия вплоть до аварии. Критерием эффективности ОС в этом случае является способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата. Это время называется временем реакции системы, а соответствующее свойство – реактивностью.

Интересная форма мультипрограммной работы связана с *мультипроцессорной обработкой*. Мультипроцессорная обработка – это способ организации вычислительного процесса в системе с несколькими процессорами, при котором несколько задач (процессов, потоков) могут одновременно выполняться на разных процессорах системы. Концепция мультипроцессирования не нова, она известна с 70-х годов, однако стала доступной в широком масштабе лишь в последнее десятилетие, особенно с появлением многопроцессорных ПК (часто в качестве серверов ЛВС).

Управление процессами и потоками

Процесс – это программный модуль, выполняемый в центральном процессоре (*CPU*). В обычных ОС процесс появляется при запуске какой-нибудь программы. Он всегда находится в активном состоянии. В ОС реального времени многие процессы могут находиться в состоянии бездействия, т.е. находятся в пассивном состоянии.

ОС контролирует следующую деятельность, связанную с процессами:
• создание и удаление процессов;
• планирование процессов;
• синхронизацию процессов;
• коммуникацию процессов;
• разрешение тупиковых ситуаций.

Не следует смешивать понятия «процесс» и «программа». Программа – это план действий, а процесс – это собственно действие, поэтому понятие процесса включает:
• программный код;
• данные;
• содержимое стека;
• содержимое адресного и других регистров процессора.

Определение концепции процесса необходимо для выработки механизмов распределения и управления *ресурсами*. Понятие ресурса – это также основное понятие ОС. Ресурсом называется всякий объект, который может распределяться внутри системы.

Ресурсы бывают:

1. Делимые:
   - используемые одновременно;
   - используемые параллельно.

2. Неделимые.

При разработке первых ОС ресурсами считались:

- процессорное время;
- память;
- каналы ввода/вывода;
- периферийные устройства.

} Системные ресурсы

Таким образом, для одной программы могут быть созданы несколько процессов в том случае, если с помощью одной программы в *CPU* выполняется несколько несовпадающих последовательностей команд. За время существования процесс многократно изменяет свое состояние.

Различают следующие состояния процесса (рис. 3):

• *новый* – процесс только что создан;

• *выполняемый* (*running*) – все затребованные процессом ресурсы выделены. Команды программы выполняются в *CPU*. В таком состоянии в однопроцессорной вычислительной системе находиться только один процесс;

• *ожидающий* (*blocked*) – затребованные ресурсы не могут быть предоставлены, или процесс ожидает завершения некоторого события, чаще всего операции ввода-вывода;

• *готовый* (*ready*) – процесс ожидает освобождения *CPU*;

• *завершенный* (процесс завершил свою работу).

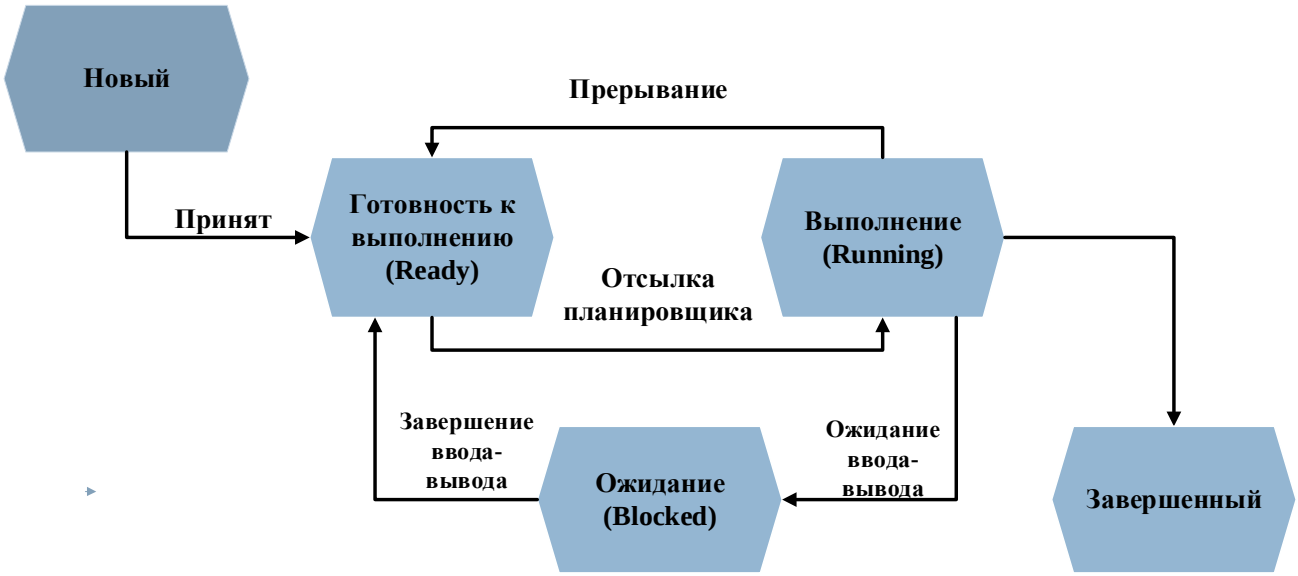


Рис. 3. Состояния процесса

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

- по команде пользователя;
- при выборе из очереди планировщиком;
- по вызову из другой задачи;
- по прерыванию от внешнего устройства, запускающего задачу;
- при наступлении запланированного времени запуска программы.

Процессу, находящемуся в состоянии готовности, выделены все необходимые ресурсы, кроме центрального процессора. При выделении процессора он перейдет в состояние выполнения.

Из состояния выполнения процесс может выйти по одной из следующих причин:

- процесс сообщает супервизору ОС о своем завершении. Супервизор либо переводит его в список бездействующих процессов (в ОСРВ), либо уничтожает (обычные ОС);
- процесс переводится супервизором ОС в состояние готовности в связи с появлением более приоритетной задачи или в связи с окончанием выделенного ему кванта времени;
- процесс блокируется либо вследствие запроса операции ввода/вывода, либо в силу невозможности предоставить запрошенный ресурс.

При наступлении соответствующего события (завершилась операция ввода/вывода, освободился затребованный ресурс, в оперативную память загружена необходимая страница виртуальной памяти) процесс деблокируется и переводится в состояние готовности.

Таким образом, движущей силой, меняющей состояние процессов, являются события. Один из основных видов событий – это прерывания.

На рис. 3 приведена типовая диаграмма переходов для состояний процессов. Каждый процесс представлен в ОС набором данных, называемых *таблицей управления процессом* (ТУП - *PCB – process control block*). В *PCB* процесс описывается набором значений, параметров, характеризующих его текущее состояние и используемых ОС для управления прохождением процесса через компьютер.

Основные черты мультипрограммного режима:

- В оперативной памяти находятся несколько пользовательских программ в состояниях *активности*, *ожидания* или *готовности*.
- Время работы процессора разделяется между программами, находящимися в памяти в состоянии готовности.
- Параллельно с работой процессора происходит подготовка и обмен с несколькими устройствами ввода-вывода.

Мультипрограммирование предназначено для повышения пропускной *способности* вычислительной системы путем более равномерной и полной загрузки всего ее оборудования, в первую *очередь* процессора. При этом скорость работы самого процессора и номинальная *производительность* ЭВМ не зависят от *мультипрограммирования*.

Мультипрограммный режим имеет в ЭВМ аппаратную и программную поддержку:

аппаратная:

- *контроллеры устройств* ввода-вывода, которые могут работать параллельно с процессором;
- система прерывания;
- аппаратные средства системы защиты программ и данных в микропроцессоре;
- и т.п.;

программная:

- мультизадачная операционная система (ОС);
- системные программы, управляющие работой устройств ввода-вывода и специализированных средств вычислительной системы.

Управляющая *программа* (ОС), реализуя *мультипрограммный режим*, должна
распределять (в том числе динамически) *ресурсы* системы (время процессора, оперативную и внешнюю *память*, устройства ввода-вывода и т.д.) между параллельно выполняемыми программами, чтобы обеспечить увеличение *пропускной способности* компьютера с учетом ограничений на *ресурсы* и требований *по* срочности выполнения отдельных про грамм.

Производительность мультипрограммной ЭВМ оценивается количеством задач, выполненных в единицу времени *(пропускная способность)* и *временем выполнения каждой программы*.

Основные классификации операционных систем

Операционные системы могут различаться особенностями реализаций внутренних алгоритмов управления основными ресурсами компьютера (процессорами, устройствами, памятью), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

Существует несколько классификаций операционных систем, в которых выделяют определенные критерии, отражающие разные существенные характеристики систем, рассмотрим наиболее часто встречающиеся:

По назначению

1. Системы общего назначения.

Подразумевает ОС, предназначенные для решения широкого круга задач, включая запуск различных приложений, разработку и отладку программ, работу с сетью и мультимедиа.

2. Системы реального времени.

Предназначены для работы в контуре управления объектами.

3. Прочие специализированные системы.

Это различные ОС, ориентированные, прежде всего на эффективное решение определенного класса, с большим или меньшим ущербом для прочих задач.

По характеру взаимодействия с пользователем

1.Пакетные ос, обрабатывающие заранее подготовленные задания.

2.Диалоговые ОС, выполняющие задания пользователя в интерактивном режиме.

*3.*ОС с *графическим интерфейсом.*

4.Встроенные ОС, не взаимодействующие с *пользователем.*

По числу одновременного выполнения задач

1. Однозадачные.

В таких системах в каждый момент времени может существовать не более чем один пользовательский процесс. Однако, одновременно с этим, могут работать системные процессы.

2. Многозадачные.

Они обеспечивают параллельное выполнение которых пользовательских процессов. Реализация многозадачности требует значительного усложнения алгоритмов и структур данных, используемых в системе.

По числу одновременных пользователей

1. Однопользовательские

Для них характерен полный пользовательский доступ к ресурсам. Подобные системы приемлемы в основном на изолированных компьютерах.

2. Многопользовательские

Их важной компонентой являются средства защиты данных и процессов каждого пользователя, основанные на понятии владельца ресурса и на точном указании прав доступа, предоставленных каждому пользователю системы.

По аппаратурной основе

1. Однопроцессорные

2. *Многопроцессорные*

В задачи такой системы входит эффективное распределение, выполняемых заданий по процессорам и организация согласованной работы всех процессоров.

3. Сетевые

Они включают возможность доступа к другим компьютерам локальной сети, работы с файловыми и другими серверами.

4. Распределенные.

Распределенная система, используя ресурсы локальной сети, представляет их пользователю как единую систему, не разделенную на отдельные машины.

По способу построения

1.Микроядерные.

2.Монолитные.

Классификация операционных систем по семействам

Операционные системы семейства OS/2

OS/2 – семейство многозадачных операционных систем с графическим интерфейсом, есть версии для многопроцессорных машин.

OS/2 создавалась для собственных нужд и задач фирмы *lBM.*

OS/2 использовалась *IBM* в качестве основы некоторого числа программных решений, таких как комментаторские системы олимпийских игр, программное обеспечение для банков. Под нее практически не существует программного обеспечения.

Поддержка OS/2 до последнего времени осуществлялась выпуском версий OS/2 безо всяких кардинальных изменений и улучшений.

Исторически сложилось такая ситуация, что в данный момент эта ОС на рынке программного обеспечения мало распространена. Существует несколько версий ОС *OS/2 WarpServer*, являющихся операционными системами для серверов.

В рамках проекта *Core/2* существуют два действующих направления по развитию *OS/2:*

• *OS/4* - создание современного ядра методом реверс-инижиринга и, полного переписывания кода на основе существующих ядер.

• OS Free – создание всей операционной системы «с нуля» на основе современных микроядерных технологий и активного использования OpenSource наработок.

Операционные системы семейства UNIX

Первая система *UNIX* была разработана в 1969 г. в подразделении *Bell Labs* компании *AT & T.* С тех пор было создано большое количество различных *UNIX-систем.* Все ОС, относящиеся к этому семейству, являются многозадачными, многопользовательскими, с графическим интерфейсом,

обеспечивают достаточную надежность и защиту данных. Эти ОС ставятся на различные аппаратные платформы (как на ПК, так и на большие машины такие как мэйнфреймы и суперЭВМ).

Некоторые отличительные признаки *UNIX*-систем включают в себя:

• использование простых текстовых файлов для настройки и управление системой;

• широкое применение утилит, запускаемых в командной строке;

• взаимодействие с пользователем посредством виртуального устройства - терминалом;

• использование конвейеров из нескольких про грамм, каждая из которых выполняет одну задачу;

• предоставление физических и виртуальных устройств и некоторых средств межпроцессорного взаимодействия как файлов.

Идеи, заложенные в основу *UNIX,* оказали огромное влияние на развитие компьютерных операционных систем. В настоящее время *UNIX-системы* признаны одними из самых исторически важных ОС.

Совокупная доля различных *UNIX*-систем занимает значительную долю на рынке серверных программ. Ввиду большой надежности системы *UNIX* она широко используется для организации работы глобальной сети Intemet.

Операционные системы семейства Linux

Linux является одной из распространенных систем версий *UNIX*. Она может организовывать работу как рабочих станций, так и сервера. Поддерживает технологию *Plug & Play* (стандарт аппаратной и программной архитектуры, который делает возможным распознавание устройств).

Linux – это многозадачная и многопользовательская операционная система для бизнеса, образования и индивидуального программирования. Как и все *UNIX-системы*, она ориентирована на работу в сети.

Одним из достоинств *Linux* можно считать высокую скорость работы. Эта ОС может работать на машинах не очень большой мощности. Второе достоинство заключается в том, что она может применяться как для различных типов серверов, так и для настольных компьютеров.

В отличие от большинства других операционных систем, *Linux* не имеет единой «официальной» комплектации. Вместо этого *Linux* поставляется в большом количестве так называемых дистрибутивов, в которых ядро *Linux* соединяется с утилитами *GNU* и другими прикладными программами (например, *X.org),* делающими её полноценной многофункциональной операционной средой.

Операционные системы семейства Windows

История Windows началась в 1985 году, когда появилась первая версия системы (оболочки). Через несколько лет вышла вторая версия, но особой популярности система Windows не завоевала.

В 1990 году вышла **Windows 3.0,** которая стала применяться на многих ПК (графический интерфейс, многозадачный режим, появление множества программ, работающих под управлением Windows).

Последующие версии **Windows** были направлены на повышение надежности, на поддержку средств мультимедиа и работу в компьютерных сетях.

Всех представителей **ОС Windows** можно разделить на две линейки:

1. Windows 9.x (95/98/Me).
2. Windows NT (NT4/2000/XP/2003 Server/Vista/2008 Server/7).

Только в семействе Windows NT представлены операционные системы для серверов.

ОС семейства Windows обладают следующими **характерными особенностями**:

- Многопользовательские ОС.
- Многозадачные ОС.
- Сетевые и несетевые ОС.
- Графические ОС.
- 32/64-разрядные.
- Подключение новых устройств по технологии Plug and Play.
- Файловая система: FAT32, NTFS.

Представители семейства Windows:

Windows 3.x (3.0/3.1/3.11). Операционные оболочки, выполняемые под управлением MS-DOS.

Windows 95 (первая ОС). Изменился интерфейс, выросла скорость работы программ, возможность автоматической настройки дополнительного оборудования, возможность работы с Интернет.

Windows 95 OSR2. Исправлены многие ошибки Windows 95, добавлена поддержка нескольких новых устройств, возможность использовать файловую систему FAT32.

Windows 98. Сохранился внешний интерфейс, переработана внутренняя структура, много внимания уделено работе с Интернет, возможность работы с несколькими мониторами.

Windows 98 SE. В состав включена 5-ая версия Internet Explorer, обновленная система соединения с Интернет, многочисленные исправления ошибок и новая библиотека драйверов.

Windows NT (1992г. — NT 3.0, 1994г. — NT 3.5, 1996г. — NT 4.0), разрабатывались с целью повышения надежности и мощности сетевой работы. Выпускается в двух модификациях:

1. Windows NT Server – предназначена для управления сетевыми ресурсами.

2. Windows NT Workstation – предназначена для работы на локальных компьютерах и рабочих станциях**.**

Windows 2000 (NT 5.0). Разработана на основе Windows NT и унаследовала от нее высокую надежность и защищенность информации от постороннего вмешательства.

Windows Me. Наследница Windows 98, приобрела новые возможности: улучшенная работа с мультимедиа, возможность записи не только аудио, но и видеоинформации, мощные средства восстановления информации после сбоев.

Windows XP. Появление 64-разрядной версии, первая ОС с полностью настраиваемым интерфейсом, поддержка записи CD-R и CD-RW дисков на уровне самой ОС и др.

Windows CE. Предназначена исключительно для установки на «карманные» компьютеры.

Windows Server 2003. Содержит все функции, необходимые для серверной ОС Windows, направление на безопасность, надежность, доступность и масштабируемость. Версии: Standart Edition, Enterprise Edition, Datacenter Edition, Web Edition.

Windows Vista. План выпуска версий Windows Vista оптимизирован для ключевых категорий пользователей – отдельных пользователей, малых предприятий, средних и крупных организаций, а также для соответствующих этим категориям схем использования ПО. Основной задачей семейства Windows Vista является наиболее точное соответствие набора предлагаемых программных продуктов потребностям потребителей.

Windows Server 2008 (кодовое имя «Longhorn Server») – новая версия серверной операционной системы от Microsoft. Эта версия должна стать заменой Windows Server 2003 как представитель операционных систем поколения Vista.

Windows 7 (ранее известная под кодовыми названиями Blackcomb и Vienna) – версия компьютерной операционной системы семейства Windows, следующая за Windows Vista.

ОС семейства *Windows* реализует метод многозадачности с вытеснением. Это позволяет снять приложение в случае его зависания. Также эти ОС поддерживают технологию *OLE (Object Linking Embedding). OLE* – стандарт, позволяющий создавать различные составные документы: в документ, созданный одним приложением, можно внедрять объекты или ссылаться на те из них, которые созданы другими приложениями.

В интерфейсе ОС семейства *Windows* реализована объектная модель.

Также они поддерживают работу ПК в сети. Эта поддержка реализовывается в следующих ситуациях:

- ОС поддерживает действие машины-клиента для наиболее

распространенных серверных операционных систем;

• ОС может одновременно поддерживать различные типы машин-клиентов;

• ОС дает возможность создавать одноранговые локальные сети.

В настоящее время Microsoft Windows установлена примерно на 92% персональных компьютеров и рабочих станций. По данным компании Net Applications, в апреле 2010 года рыночная доля Windows составляла 91,5%.

ОПЕРАЦИОННАЯ СИСТЕМА WINDOWS 10

Операционная система Windows 10 появилась относительно недавно – она стала доступной с 29 июля 2015 года. Компания Microsoft при разработке продолжала свой путь, направленный на унификацию. Допускается установка на компьютеры, ноутбуки, планшеты, а также смартфоны и консоли Xbox One. Единая платформа обеспечивает возможность синхронизации настроек, как это уже было на предшествующих версиях.

Отдельного внимания заслуживает распространение операционной системы. Довольно большое количество пользователей не захочет переходить с полностью устраивающих их семерки и восьмерки. Как показывает статистика, именно они заняли значительную часть рынка. Разработчик предложил отличную возможность для пользователей данных ОС – выполнить обновление бесплатно в течение одного года с момента выпуска.

ИСТОРИЯ СОЗДАНИЯ

Следует сказать о том, почему Windows 10 не получила порядковый номер 9. Это больше связывается с маркетинговыми исследованиями. Несмотря на популярность восьмерки, Microsoft не считает её оптимальным программным продуктом. Новая версия ОС не должна была ассоциироваться с предшественниками. Другой важный момент заключается в относительно коротких сроках разработки. Это связывается с тем, что оптимальные решения во многих направления компания Microsoft уже создала ранее. Перед нею стояла задача аккумулировать все лучшее, что было ранее и добавить полезные функции.

СИСТЕМНЫЕ ТРЕБОВАНИЯ WINDOWS 10

Следует учитывать тот факт, что данная ОС может использоваться для нескольких типов устройств – от моноблоков и заканчивая мобильными телефонами или игровой приставкой Xbox One. Если говорить об основной версии для компьютеров, то имеются следующие системные требования:

- Процессор с частотой не менее 1 ГГц
- ОЗУ от 1 Гб (для 32х систем) и 2 Гб (для 64х систем)
- От 16 до 20 Гб свободного места на жестком диске
- Наличие DirectX 9 и выше

Для мобильных устройств системные требования несколько иные:

- Экран с разрешением не менее 800х480
- Оперативная память 512 Мб и выше

В целом, требования по сравнению с восьмой версией не изменились. Причина заключается в хорошей оптимизации и отсутствии принципиально новых особенностей в работе системы.

ОБНОВЛЕНИЕ ДО WINDOWS 10

Как уже было сказано ранее, маркетинговый ход компании позволяет выполнить обновление для пользователей. Оно доступно владельцам Windows 7, 8, а также 8.1. На протяжении года с момента выхода (до 29 июня 2016) обновление происходит бесплатно. Скорость скачивания зависит от конкретных условий, а сама процедура установки займет около часа. Необходимо использовать центр загрузки обновлений или непосредственно скачать ОС с официального сайта Microsoft.

Редакция Windows 10 зависит от того, какая система была ранее. Принцип установки является предельно простым. Редакция после обновления является аналогичной той, которая была на семерке или восьмерке. Если Вас это не устраивает, то придётся забыть о бесплатной акции и приобретать ОС.

ОТЛИЧИТЕЛЬНЫЕ ОСОБЕННОСТИ WINDOWS 10

С самого начала разработчики начали позиционировать систему, как возврат к основам семерки, но с многочисленными улучшениями. Учитывались преимущества восьмерки, а также запросы со стороны пользователей.

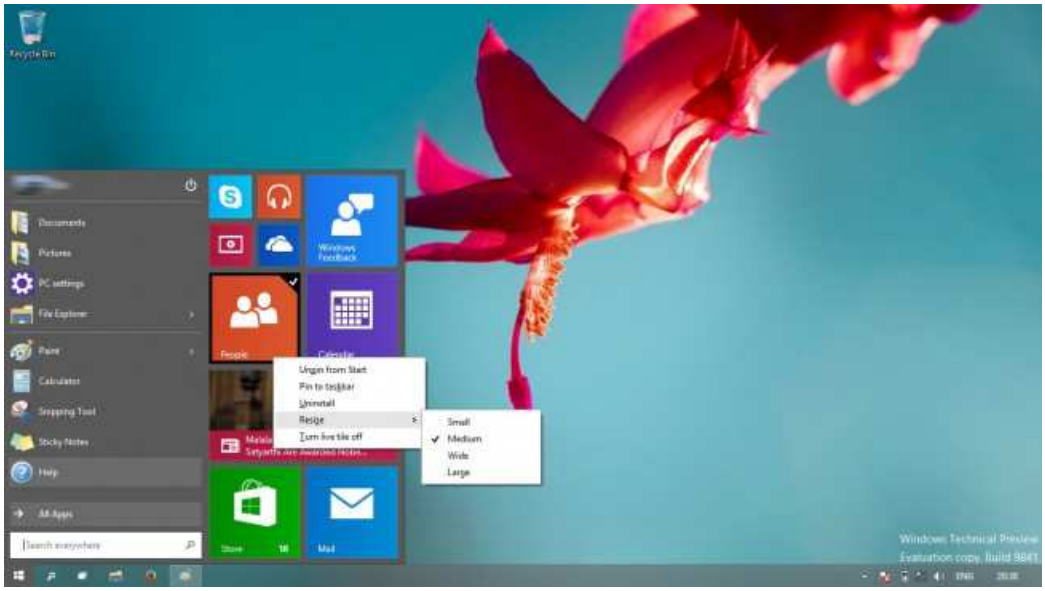
Примерно на протяжении полугода до выхода можно было принять участие в тестировании и рассказать о возможностях улучшения или доработок. Далее будут рассмотрены наиболее существенные особенности Windows 10, которые заслуживают к себе особого внимания.

Универсальность

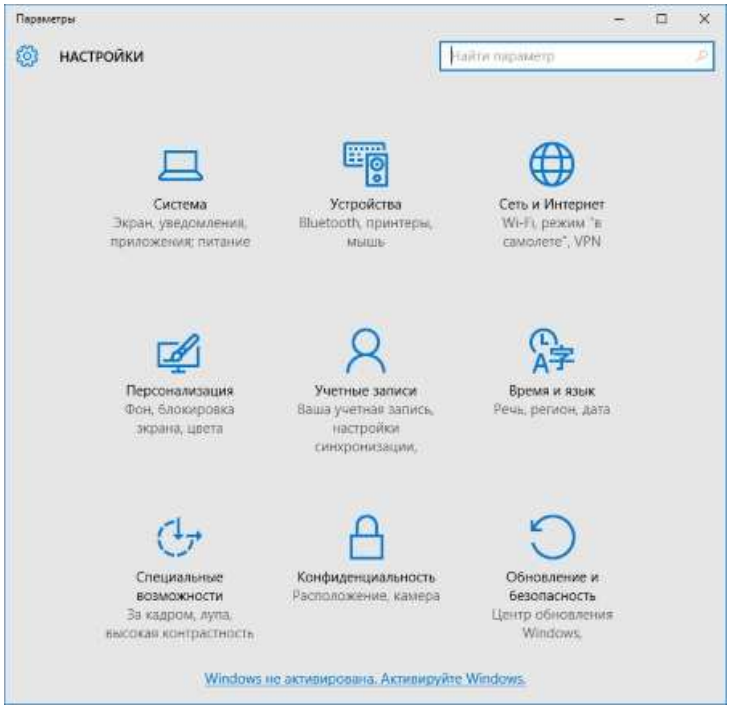
Курс на создание единой ОС продолжается и здесь компания Майкрософт немало преуспела. Серьёзные жалобы после восьмёрки шли на то, что интерфейс Metro больше подходит для планшетов и мобильных устройств, но никак не компьютеров. Разработчики приложили усилия для исправления ситуации. Универсальность прекрасна видна с промо-материалов на рекламной картинке ниже.

Улучшенный пуск

Большая часть жалоб на восьмерку относилась именно к нерациональному устранению меню «Пуск». По сути, оно было вынесено на весь рабочий стол и стало малоудобным для продвинутых пользователей. В то же время, разработчики не хотели полностью копировать семёрку в этом плане. Решение нашлось в синтезе двух вариантов. Нововведение Windows 10 заключается в том, что меню «Пуск» поделено на две части. Левая является стандартным стилем семерки, а правая больше относится к восьмой версии. Это представлено на скриншоте.

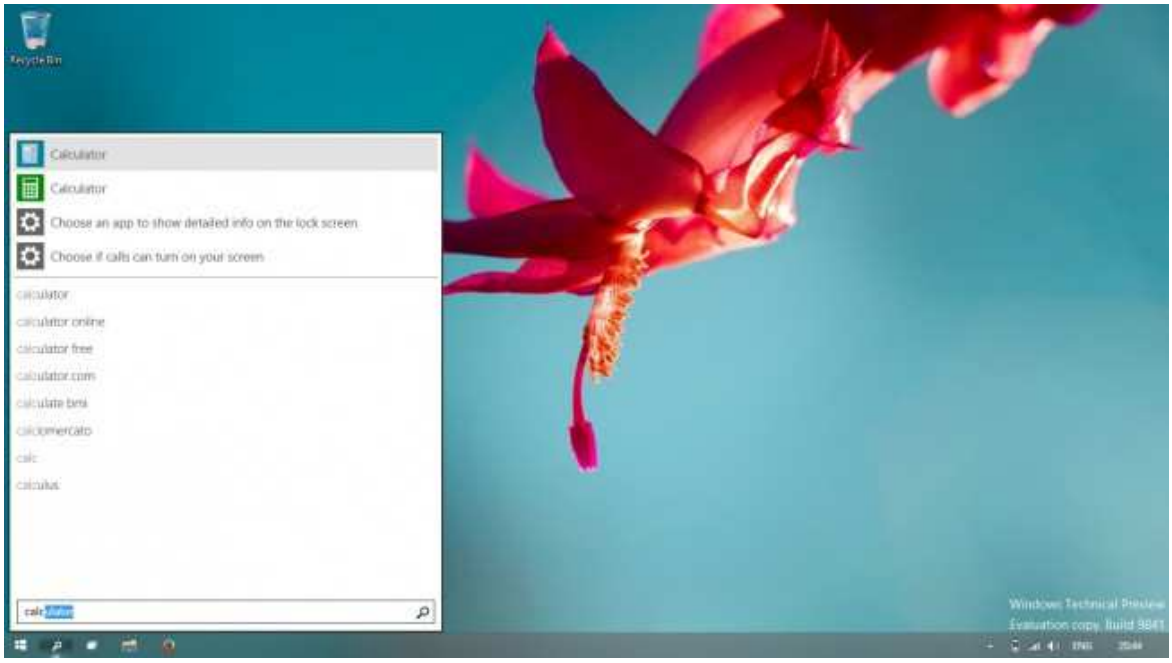


Оставлена возможность выполнить настройки по своему усмотрению. Например, вернуться с классическому интерфейсу или использовать тот, который применяется у восьмерки. Для выполнения этого требуется пройти по пути «Пуск» -> Настройки.



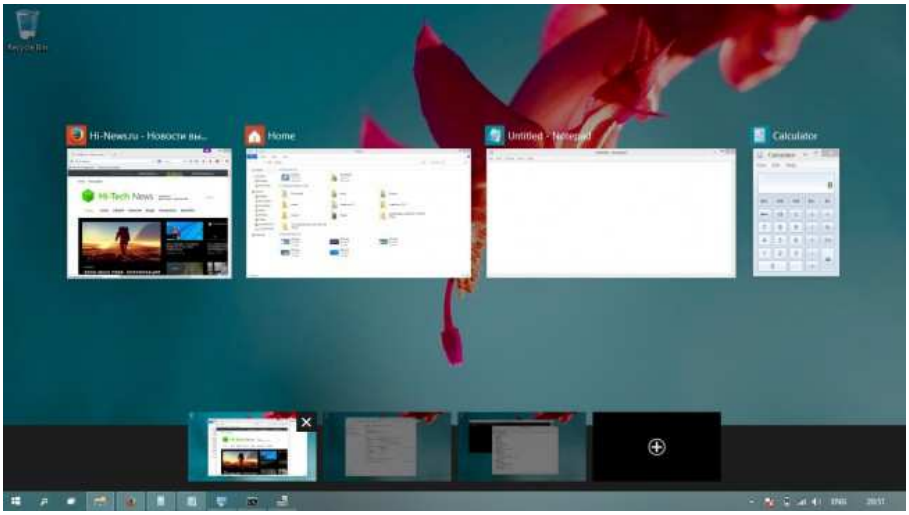
Универсальный поиск

Осуществлять поиск стало ещё проще. Теперь для его выполнения выделена отдельная кнопка, что сильно упрощает поставленную задачу. Алгоритм поиска ещё больше оптимизирован, что позволяет существенно ускорить выдачу результатов, а также потребление ресурсов в процессе.



Возможность использования виртуальных рабочих столов

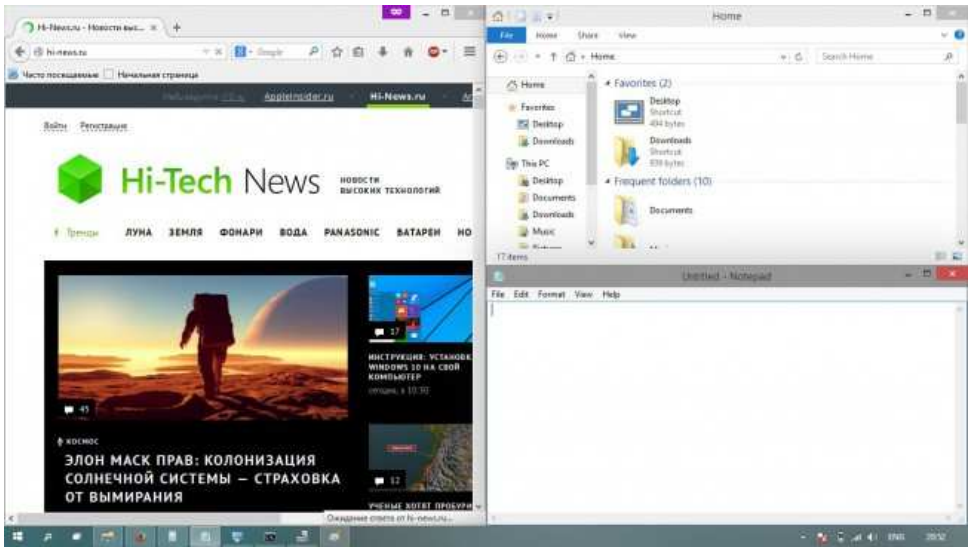
Некоторая критика программных продуктов Microsoft заключалась в том, что виртуальные рабочие столы не были реализованы в полной мере. Здесь операционная система сильно уступала большинству конкурентов. Подобная недоработка была исправлена. Имеется специальная кнопка в главном меню для вызова. Можно одновременно увидеть все рабочие столы, а также осуществлять управление ими.



Закреплена возможность выполнить быстрый вызов за счет сочетания клавиш Win+Tab. В целом, это нововведение Windows 10 можно назвать достаточно полезным.

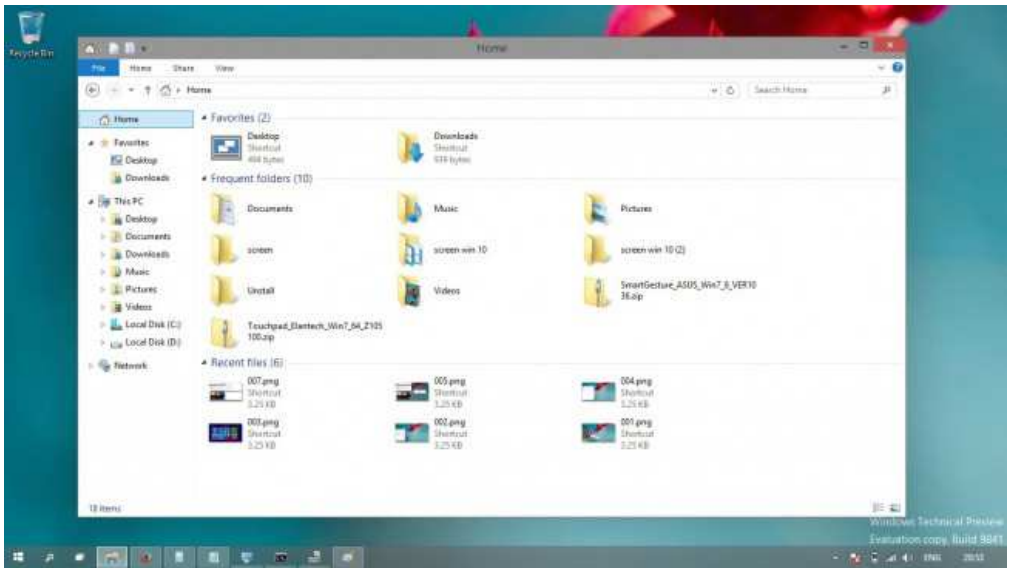
Прикрепление окон

Ещё одной доработкой стало дальнейшее развитие функции Snap. Она позволяет «крепить» окна программ на рабочий экран. В восьмерке было возможно использовать только два приложения подобным образом. Теперь данное количество увеличено до четырех.



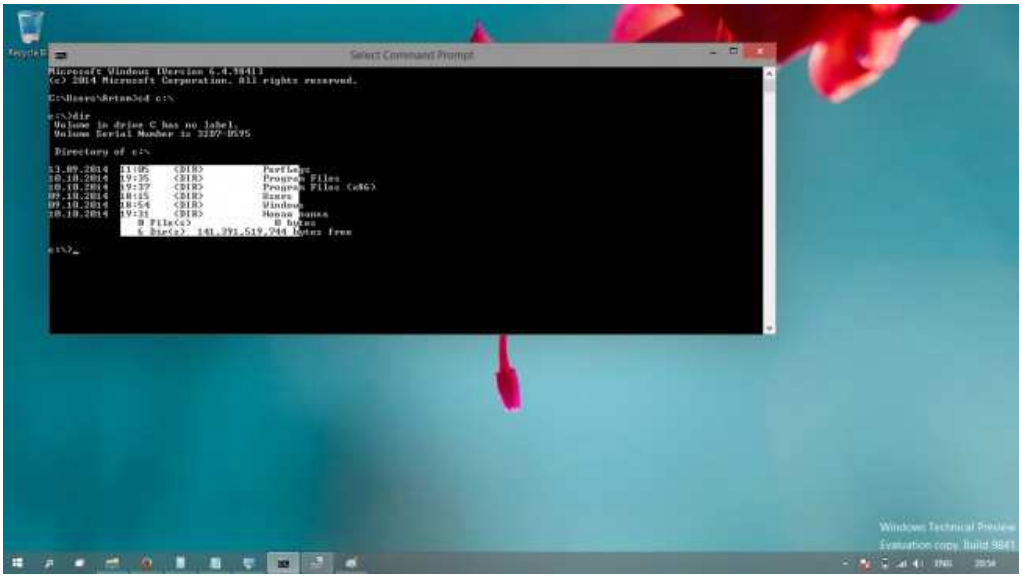
Менеджер файлов

Доступ к различным документам можно получить через менеджер файлов. Он предполагает возможность выполнить целый ряд настроек для обеспечения персонализации. Обеспечивается использование того стиля оформления, который подходит больше всего.



Улучшенная командная строка

Это улучшения является весьма значимым, но заинтересует только продвинутых пользователей. Версии операционной системы Windows меняли многие особенности работы, но командная строка всегда оставалась без изменений. Она обладала максимально упрощенным дизайном, в который не вносилось существенных корректировок со времен Windows 95. Для десятки внедрены особые возможности для работы с текстом. Сюда можно отнести выделение, вставку и копирование.

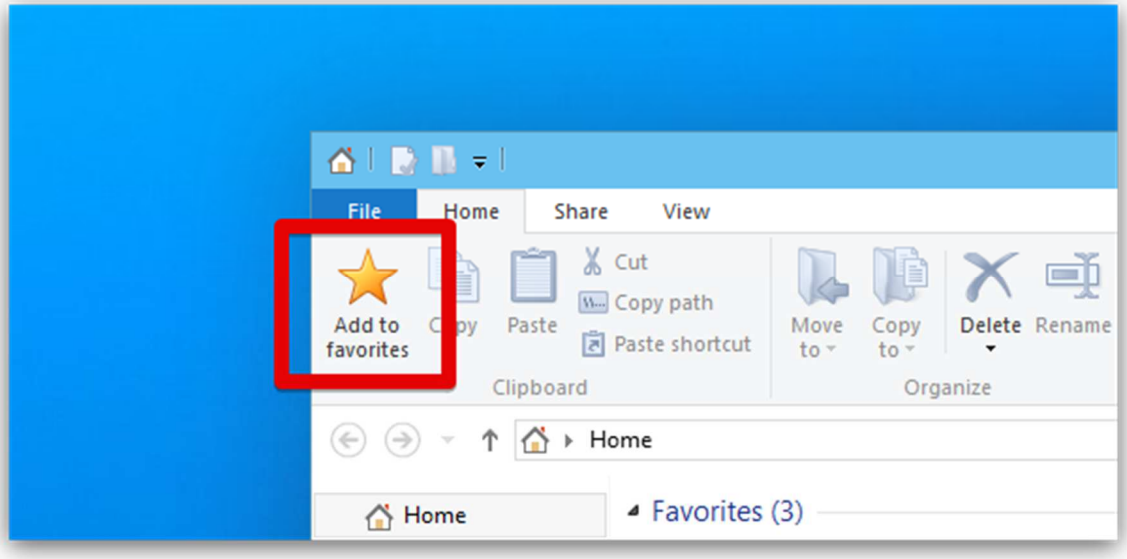


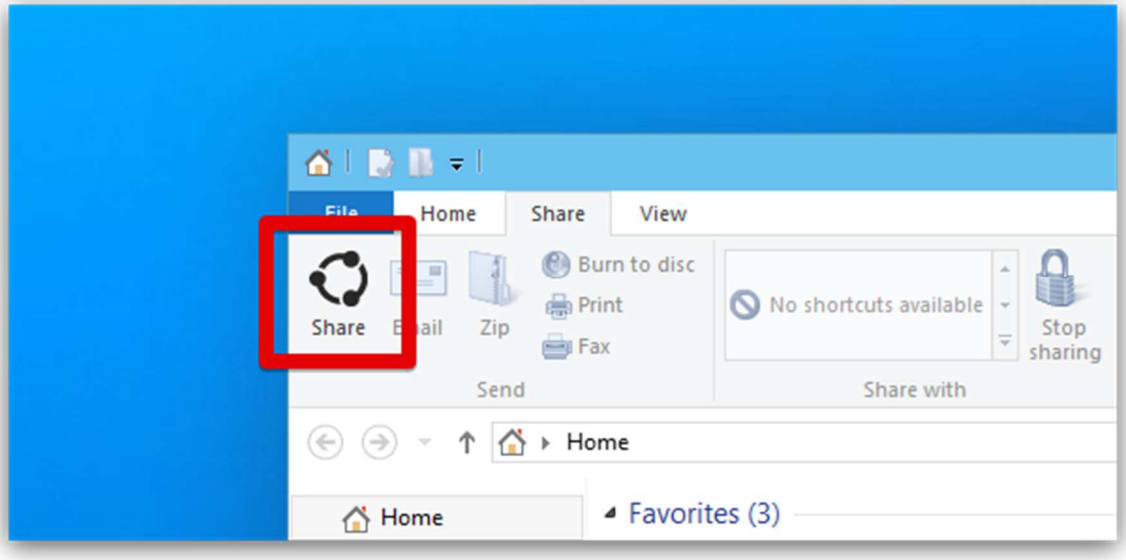
Просмотр задач

Ранее в Висте уже были реализованы приложения Flip3D и Flip. Они отличались качественной визуальной составляющей, но не были удобными. В последующих версиях подобная опция была убрана. В десятке проблема различения похожих окон убрана за счет их отрисовки с использованием более крупных элементов. Приложение Task View закрепляется непосредственно на панели задач. Это позволяет вызвать его всего одним нажатием мыши. Предусматривается тесная интеграция с функцией виртуальных рабочих столов, которая рассматривалась ранее.

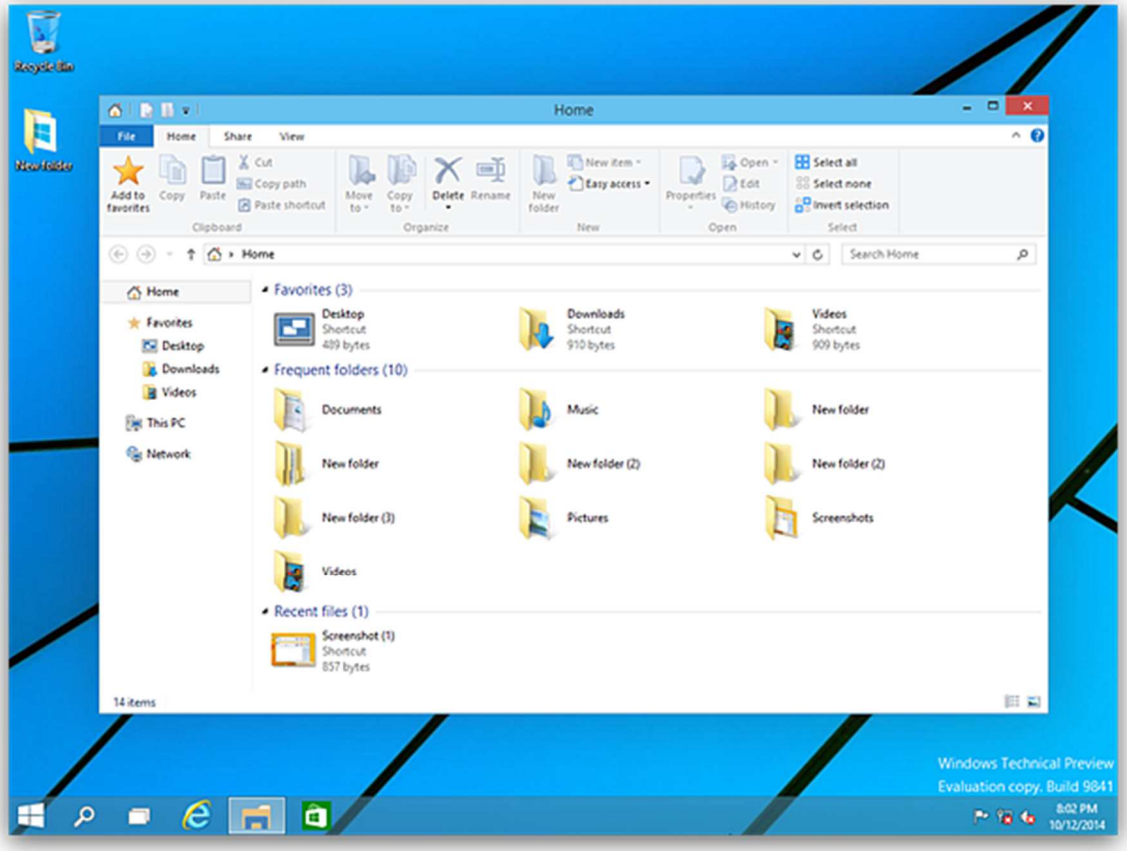
Доработки проводника

Восьмерка внедрила довольно большое количество визуальных улучшений. Как показала практика, далеко не все из них стали удачными. Это было учтено в операционной системе Windows 10. Нововведения коснулись сразу нескольких составляющих проводника. Теперь добавлено две новые кнопки: «Добавить в избранное» (помещение интересующей папки в категорию избранного) и «Поделиться» (поделиться определённым файлом/файлами со своего устройства).





Другой визуальный момент – это наличие категории «Home». Здесь находится список тех мест, которые были посещены недавно.



Другие улучшения

Есть большое количество других нововведений Windows 10, которые следует перечислить:

1. Обновленный приветственный экран и экран блокирования устройства
2. Возможность входа в систему за счет службы биометрических данных Windows Hello
3. В расширение предыдущего пункта можно сказать об использовании системы отпечатков в качестве паролей на мобильных устройствах или планшетах
4. «Панель управления» заменена «Параметрами» с более ориентированным на пользователя интерфейсом
5. Часть значков перерисована
6. Обновление часов и календаря
7. Магазин приложений Windows стал более удобным
8. Вместо Internet Explorer используется Microsoft Edge
9. Имеется новое приложение «Начало работы»

СБОР ДАННЫХ

Такой момент, как сбор данных в Windows 10 заслуживает отдельного внимания. Предусматривается системная возможность передачи различных сведений с устройства пользователя в корпорацию Microsoft. Особенность заключается в том, что количество передаваемых данных гораздо больше, чем в любой другой предшествующей ОС. Наиболее спорным моментом является конфиденциальность. Microsoft оставляет за собой право передавать полученные сведения третьим лицам, например, разработчикам ПО. Никаких признаков незаконных действий со стороны компании не предусматривается законодательством, поскольку данный момент указывается в лицензионном соглашении.

РЕДАКЦИИ WINDOWS 10

Ещё одним важным моментом, который заслуживает отдельного рассмотрения, являются редакции операционной системы. Здесь имеется 8 основных вариантов:

- **Windows 10 Домашняя.** Классическая версия, которая используется для персональных компьютеров, мобильных устройств, ноутбуков и других типов оборудования. Обладает стандартным функционалом.
- **Windows 10 Домашняя для одного языка.** По своим функциональным возможностям является полной аналогией предыдущему пункту. Главным и единственным отличием является невозможность указать другой язык.
- **Windows 10 Домашняя с Bing.** Не отличается от указанных ранее пунктов по своему функционалу, за исключением использования в браузерах Edge и Internet Explorer поисковика Bing с невозможностью изменения.
- **Windows 10 Профессиональная.** Расширенная редакция, где используется продвинутый функционал. Разработана с учетом запросов предприятий малого бизнеса.
- **Windows 10 Мобильная.** Предназначается для установки в качестве ОС на смартфоны и другие мобильные устройства (планшеты).
- **Windows 10 Корпоративная.** Редакция операционной системы для крупных компаний и организаций. Предусматривается наличие довольно больших возможностей в сфере ведения бизнеса и управления финансовыми потоками.
- **Windows 10 для образовательных учреждений.** Является некоторой доработкой профессиональной редакции.
- **Windows 10 Мобильная корпоративная.** Разновидность корпоративной редакции, предназначенная для установки на мобильные устройства.
- **Windows 10 IoT Домашняя.** Специализированная редакция, задачей которой является установка на различного вида терминалы.

КРИТИКА И НЕДОСТАТКИ WINDOWS 10

Как и у всякого крупного программного продукта, после выхода появилось большое количество критики самого разного плана. Следует рассмотреть основные моменты, вызвавшие недовольство со стороны пользователей. В первую очередь, многим не понравилась система сбора данных. Ранее уже говорилось о том, насколько большое количество сведений о пользователе она отправляет Microsoft. Сюда можно отнести местоположение, контакты и частоту разговоров с ними, данные электронной почти и другое.

Имелись и другие претензии не столь существенного плана. Они касались сложностей установки или проблем в ходе работы с некоторыми приложениями. На данный момент, "патчи" оперативно исправляют это.

Некоторая критика относится к тому, что принципиально нового подхода Windows 10 не предложила. Она использует разработки седьмой и восьмой версии. Это, действительно, так, но не стоит винить Microsoft. Семерка стала хитом и как показал опыт восьмой версии, отклонения от её стандартов способны обеспечить снижение интереса пользователей. Таким образом, происходит улучшение тех моментов, которые признаны лучшими и отказ от спорных вариантов.

ИТОГИ

Операционная система Windows 10 собрала в себе лучшее от седьмой и восьмой серии. Имеются большие возможности в плане индивидуальных настроек и красивая визуальная составляющая. Сразу после выхода ОС можно было назвать «сырой», но сейчас это уже исправлено. Все говорит в пользу того, что на протяжении следующих лет она войдет в число наиболее популярных.

Архитектура аппаратных и программных средств персонального компьютера

В самом общем виде аппаратные средства персонального компьютера можно представить в виде нескольких блоков (рис.1).

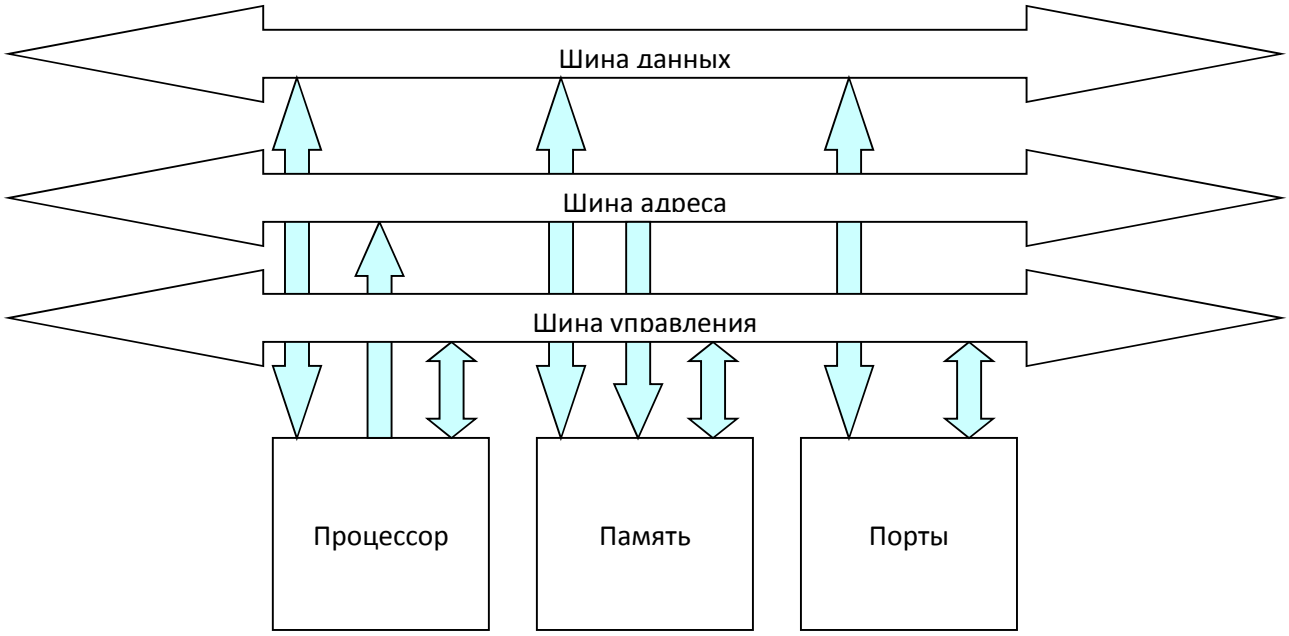


Рис.1. Аппаратные средства персонального компьютера

Шина данных используется для переноса информации между основными узлами компьютера. Перенос данных происходит по параллельным линиям, их количество называют шириной шины.

Шиной адреса управляет микропроцессор.

Память может быть односторонней (чтение) – ПЗУ (ROM) хранит программы и данные, образующие в совокупности базовую систему ввода/вывода BIOS и двухсторонней (чтение и запись) – оперативная память (RAM).

Порты – специальные аппаратные регистры, используемые для управления аппаратными средствами ПК. Каждый порт имеет свой уникальный *номер*.

На *шине управления* микропроцессор выставляет команды управления узлами системы и получает ответные сигналы состояния узлов и подтверждение выполнения команды.

Ширина шины данных и ширина шины адреса являются важнейшими характеристиками микропроцессора.

Микроархитектура процессоров 8086

Первым представителем семейства Intel *x*86, или, согласно официальной классификации фирмы *Intel (Integrated Electronics*, США), семейства процессоров *IA* (*Intel Architecture*), является микропроцессор 8086, выпущенный 8 июня 1978 года. Реализованная в процессоре архитектура набора команд стала основой широко известной архитектуры *x*86. Процессоры этой архитектуры стали наиболее успешной линией процессоров *Intel*. Современные процессоры этой архитектуры сохраняют возможность выполнять все команды этого набора. Незначительно изменённая версия процессора с 8-битной шиной данных, выпущенная в 1979 году под названием *Intel* 8088, применялась в персональных компьютерах *IBM PC и IBM PC/XT*.

Сегодня процессоры этого семейства стали стандартом де-факто для большинства персональных компьютеров (ПК) во всем мире.

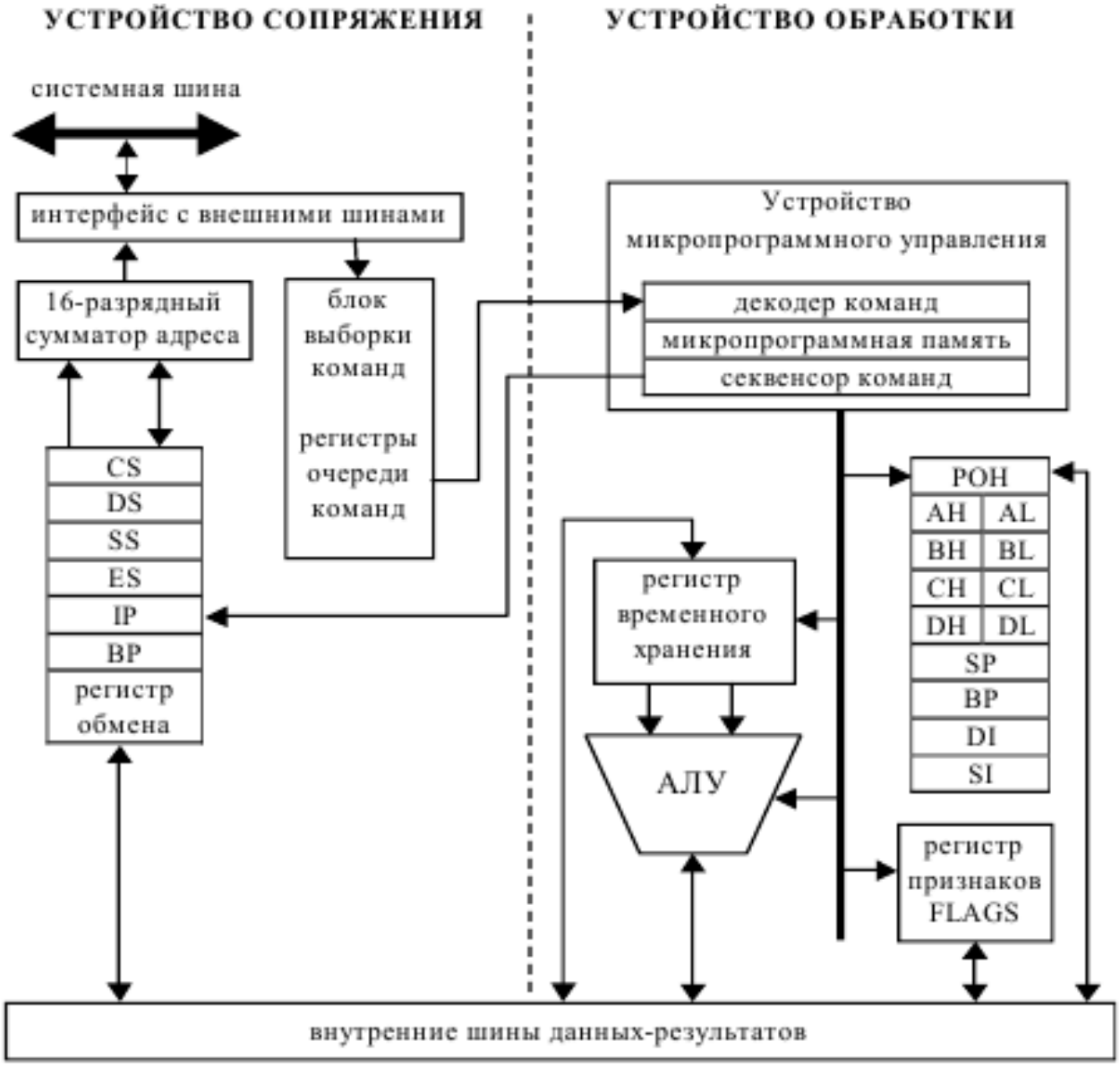
Микропроцессоры *Intel* 8086/88 имеют 20-разрядную шину адреса и 16/8-разрядную шину данных. Ширина шины адреса устанавливает для ПК *IBM PS XT*, основанного на микропроцессоре *i*8088, ограничение на объем адресуемой памяти в 1 Мбайт (2^{20} байт).

Микропроцессоры *Intel* 8086/88 работают в *реальном режиме* (*real mode*):

- Адрес памяти, формируемый микропроцессором, является физически адресом;
- все машинные команды системы команд микропроцессора разрешены для исполнения любой программой;
- объем адресуемой памяти ≤ 1Мбайт.

Микропроцессор 8086 ориентирован на выполнение команд параллельно с их выборкой и может быть условно разделен на две части, работающие асинхронно (рис. 2): устройство сопряжения с внешними шинами (УС) и устройство обработки (УО). Устройство сопряжения обеспечивает формирование 20-разрядного физического адреса памяти, выборку команд и операндов из памяти, организацию очередности команд и запоминание результатов выполнения команд в памяти. В состав УС входит шесть 8-разрядных регистров очереди команд, четыре 16-разрядных сегментных регистра, 16-разрядный регистр обмена и 16-разрядный сумматор адреса, интерфейс с внешними шинами. Регистры очереди команд организованы по принципу *FIFO* - «первым пришел - первым вышел». УС готово выполнить цикл выборки 16-разрядного слова из памяти всякий раз, когда в очереди освобождаются, по меньшей мере, два байта, а УО извлекает из очереди команды по мере их выполнения. При выполнении команд передачи управления, например, условных и безусловных переходов, очередь очищается УС и начинает заполняться заново.

Устройство обработки предназначено для выполнения операций по обработке данных и состоит из устройства микропрограммного управления (УМУ), 16-разрядного АЛУ, восьми 16-разрядных регистров общего назначения и регистра признаков. Команды из очереди, сформированной УС, поступают в УМУ, где декодируются и выполняются в 16-разрядном АЛУ согласно процедурам, записанным в памяти микропрограмм. Последовательное выполнение команд обеспечивается секвенсором команд, часть которого (регистр счетчика команд *IP*) изображена в составе УС, т.к. именно УС записывает в *IP* смещение следующей команды, т.е. положение новой команды относительно начала сегмента команд. УО обменивается данными с УС через внутреннюю 16-разрядную шину и регистр обмена.



. Рис. 2. Микроархитектура процессора 8086

Среди программно-доступных регистров выделяют следующие группы (рис. 3):

- Регистры данных: *AX* – аккумулятор (*Accumulator*); *BX* – базовый регистр (*Base*); *CX* – регистр счетчика (*Counter*); *DX* – регистр данных (*Data*).
- Регистры-указатели (индексные регистры): *SI* – индекс источника (*Source Index*); *DI* – индекс приемника (*Destination Index*); *BP* – указатель базы (*Base Pointer*); *SP* – указатель стека (*Stack Pointer*).
- Сегментные регистры: *SS* – сегмент стека (*Stack Segment*); *DS* – сегмент данных (*Data Segment*); *ES*–дополнительный сегмент (*Extended data Segment*); *CS* – сегмент кода (*Code Segment*).

16-битные регистры *AX, BX, CX, DX* состоят из двух 8-битных половин, к которым можно независимо обращаться по именам *AH, BH, CH, DH* – старшие байты и *AL, BL, CL, DL* - младшие байты.

Рис. 3. Регистры процессора 8086

Регистр флагов или слово состояния процессора (*PSW*) содержит 16 бит, из которых используется только 9 (рис. 4).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рис. 4. Регистр флагов

Флаги условий устанавливаются аппаратурой арифметико-логического устройства микропроцессора по результатам выполнения машинной команды:

- *CF* – флаг переноса;

- *PF* – флаг паритета;
- *AF* – флаг дополнительного переноса;
- *ZF* –флаг нуля;
- *SF* – флаг знака;
- *OF* – флаг переполнения.

Флаги управления влияют на функционирование аппаратуры процессора:

- TF – флаг трассировки;
- DF – флаг направления;
- IF – флаг управления маскируемыми прерываниями (1- процессор реагирует на прерывания, генерируемые внешними устройствами, 0 – не реагирует).

Формирование физического адреса

Так как внутренние регистры – 16-разрядные (2^{16} = 64 Кбайт), а шина адреса имеет ширину 20 бит (2^{20} = 1Мбайт), то для формирования 20-разрядного физического адреса памяти используются: *сегмент адреса и смещение адреса*. Физический адрес записывается парой этих значений, разделенных двоеточием, и образуется следующим образом (рис. 5):

- значение сегмента адреса, берущегося из регистра, *CS* сдвигается на 4 бита влево с заполнением разрядов справа нулями;
- к образовавшемуся 20-битовому значению прибавляется значение смещения адреса из регистра *IP*. При возникновении переполнения берется только пять 16-ричных цифр результата.

Например, адрес 40:1*Ch* соответствует физическому адресу 0041*Ch*.

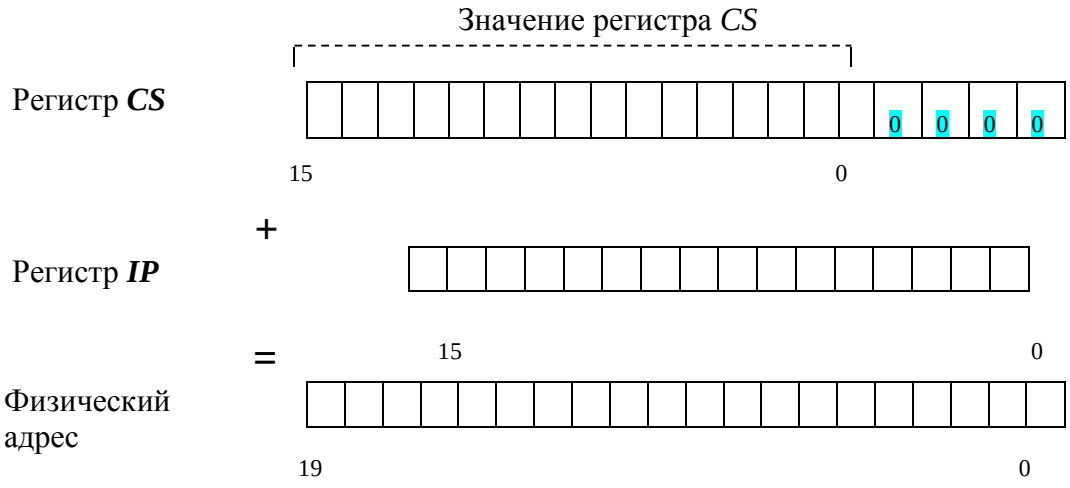


Рис. 5. Схема определения физического адреса для микропроцессоров 8086/88

Память организована в виде одномерного массива байтов с физическими адресами от 00000_{16} до $FFFFF_{16}$. Две области адресного пространства памяти зарезервированы для выполнения специальных функций, связанных с обработкой прерываний и системным сбросом. Этими областями являются первые 128 байт (физические адреса 00000 - 0007*F*) и последние 16 байт (физические адреса *FFFF*0 - *FFFFF*). Данные области использовать для других целей нельзя. Байты в памяти организуются в слова таким образом, что байту, имеющему меньший адрес, соответствуют менее значимые позиции разрядов в слове. Каждый байт или слово памяти адресуется с помощью 20-битного адреса, причем в случае адресации слова адрес указывает на его младшую часть. Например, адрес 00000_{16} может обозначать и байт с этим адресом, что условно записывается в виде [00000] = 34*h*, и слово с таким же адресом, что записывается в виде [00000] = 1234*h*. Тогда старший байт слова [00001] = 12*h*. Квадратные скобки обозначают ячейку памяти, адрес которой находится в этих скобках, *h* – шестнадцатеричную систему счисления.

Прерывания

Микропроцессоры Intel 8086/88 поддерживают 256 прерываний. Каждое из них имеет свой номер и *ISR*. Адрес точки входа в *ISR* называется вектором *прерываний* и хранится в специальной таблице, называемой *таблицей векторов прерывания (ТВП)*. Код *ISR* может располагаться в любом месте памяти. Поэтому вектор прерывания занимает 4 байта:

- 2 байта отводится на значение сегмента адреса – значение регистра *CS*;
- 2 байта – на значение смещения, устанавливаемое в регистре *IP*.

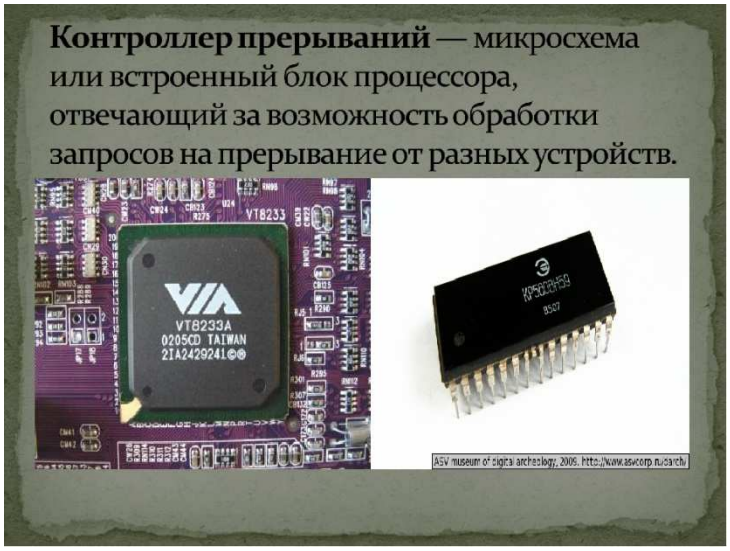
Вся ТВП занимает 256*4 = 1024 байта и хранится в оперативной памяти, начиная с адреса 0000:0000.

При возникновении прерывания процессор помещает в стек 6 байт:

- текущее значение регистра *CS*;
- текущее значение регистра *IP*;
- 2 байта флагов процессора;
- в *CS* и *IP* устанавливаются значения из ТВП, задающие начальный адрес ISR.

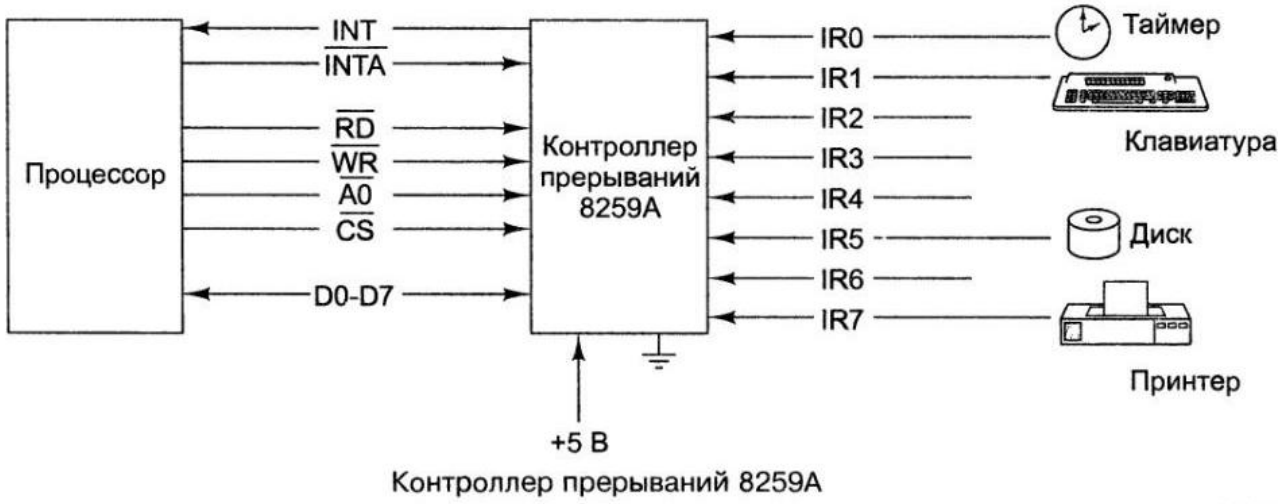
Прерыванию «0» соответствует вектор прерывания по адресу 0000:0000.
Прерыванию «1» соответствует вектор прерывания по адресу 0000:0004*h*.
Прерыванию «2» соответствует вектор прерывания по адресу 0000:00008*h*.
……..
В процессе функционирования ПК могут встретиться 4 типа прерываний:

1. *Аппаратные*.



Они возникают как результат внешних событий, и их генерирует специальная микросхема ПК *i*8259*A* – *программируемый контроллер прерываний* (*PIC*).

Она рассчитана на 8 входов запросов прерываний (*IR* – ***I****nterrupt* ***R****equest*). Сигналы на ней возбуждают внешние устройства: таймер, клавиатура, дисководы и т.д. Восприняв запрос на одном из этих входов *i*8259*A*, формирует сигнал на выходе *INT*. Выход этой микросхемы подается на специальный вход процессора *INTR*. Этот вход является маскируемым: если флаг *IF* равен 1, то процессор реагирует на прерывания по этой линии, *флаг IF* сброшен в 0, то – нет;



Контроллер прерываний 8259A

Микропроцессор подтверждает прерывание, выставляя два сигнала на выходе *INTA*. Эти сигналы поступают на одноименный вход контроллера, и по второму из них *i*8259*A* выставляет на шину данных тип прерывания.

Внутренняя структура программируемого контроллера прерываний *i*8259*A* представлена на рис. 6.

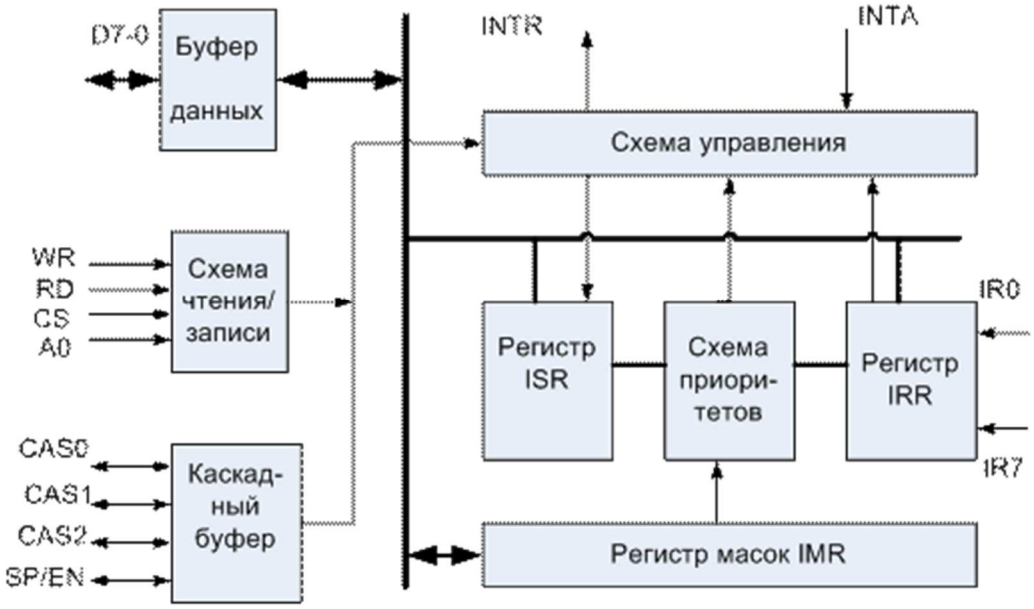


Рис. 6. Внутренняя структура *i*8259*A*

Буфер данных используется для связи контроллера с шиной данных процессора. Контроллер подключается к шине данных, когда на его вход *CS* приходит активный (нулевой) сигнал и при подтверждении прерывания (сигнал *INTA*).

Схема чтения/записи управляет чтением/записью внутренних регистров контроллера.

Каскадный буфер содержит схему распознавания каскадного соединения нескольких контроллеров. Каскадное соединение 2-х контроллеров позволяет организовать 15 входов запросов на прерывание.

IRR – восьмиразрядный регистр запросов прерываний. В нем запоминаются все запросы, приходящие на входы *IR*7 – *IR*0.

ISR – восьмиразрядный регистр обслуживаемых запросов.

2. *Немаскируемые прерывания.*

Процессор, кроме входа *INTR,* имеет вход немаскируемых прерываний *NMI*. Сигнал по этому входу не может быть блокирован программным образом и вызывает прерывание с номером «2». Это прерывание имеет более высокий приоритет, чем по входу *INTR* и используется для организации реакции процессора на критические ситуации.

3. *Программные прерывания.*

Осуществляется по программной инструкции *INT* («номер прерывания»).

4. *Исключительные ситуации.*

Генерация внутренних прерываний процессором при возникновении необычных условий при выполнении машинных команд, например, «деление на нуль», «пошаговое исполнение

Аппаратная поддержка мультипрограммирования на примере процессора *Pentium*

Аппаратные средства поддержки мультипрограммирования имеются во всех современных процессорах. Несмотря на различия в реализации, для большинства типов процессоров эти средства имеют общие черты. Более того, средства поддержки операционной системы во всех этих процессорах построены почти идентично, поэтому далее в тексте для их обозначения используется обобщенный термин «процессоры *Pentium*».

Основным режимом работы процессора *Pentium* является защищенный режим (*protected mode*). Для совместимости с программным обеспечением, разработанным для предшествующих моделей процессоров *Intel* (главным образом, модели 8086), в процессорах *Pentium* предусмотрен так называемый реальный режим (*real mode*) . В реальном режиме процессор *Pentium* выполняет 16-разрядные инструкции и адресует 1 Мбайт физической памяти. Здесь будем рассматривать защищенный режим работы процессора, поскольку это основной режим, используемый современными мультипрограммными операционными системами.

Регистры процессора

В организации вычислительного процесса важную роль играют регистры процессора. В процессорах *Pentium* эти регистры делятся на несколько групп (рис.1):

- регистры общего назначения;
- регистры сегментов;
- указатель инструкций;
- регистр флагов;
- управляющие регистры;
- регистры системных адресов;
- регистры отладки и тестирования, а также регистры математического сопроцессора, выполняющего операции с плавающей точкой.

Регистры общего назначения привлекаются для хранения:

- операндов команд целочисленного устройства;
- адресов и компонентов адреса.

Перечислим регистры, относящиеся к группе регистров общего назначения и физически находящиеся в процессоре внутри арифметико-логического устройства (регистры АЛУ):

- регистр-аккумулятор (*accumulator register*) *EAX/AX/AH/AL* применяется для хранения промежуточных данных, в некоторых командах его использование обязательно;

- базовый регистр (*base register*) *EBX/BX/BH/BL* задуман как место хранения базового адреса некоторого объекта в памяти;
- регистр-счетчик (*count register*) *ECX/CX/CH/CL* применяется в командах, производящих некоторые многократные действия;
- регистр данных (*data register*) *EDX/DX/DH/DL*, так же, как и регистр *EAX/AX/AH/AL*, хранит промежуточные данные.
- регистр индекса источника (*source index register*) *ESI/SI* в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;
- регистр индекса приемника (*destination index register*) *EDI/DI* в цепочечных операциях содержит текущий адрес в цепочке-приемнике.
- регистр указателя стека (*stack pointer register*) *ESP/SP* содержит указатель на вершину стека в текущем сегменте стека;
- регистр указателя базы кадра стека (*base pointer register*) *EBP/BP* предназначен для организации произвольного доступа к данным внутри стека.

Рис. 1**.** Основные регистры процессора *Pentium*

Процессоры *Pentium* аппаратно поддерживают сегментную организацию программы. Для этого в них имеется ***шесть сегментных регистров*** *CS, SS, DS, ES, GS, FS***,** служащих для доступа к четырем типам сегментов.

- *Сегмент кода*. Содержит команды программы. Для доступа к этому сегменту служит регистр сегмента кода (*code segment register*) *CS*.

- *Сегмент данных*. Хранит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр сегмента данных (*data segment register*) *DS*, в который помещен адрес сегмента данных текущей программы.
- *Сегмент стека*. Представляет собой область памяти, называемую стеком. Работу со стеком процессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Доступ к области стека выполняется через регистр сегмента стека (*stack segment register*) *SS*.
- *Дополнительный сегмент данных*. Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в регистре сегмента данных *DS*. Если программе недостаточно одного сегмента данных, то она имеет возможность одновременно задействовать еще три дополнительных сегмента данных, адреса которых должны содержаться в регистрах дополнительного сегмента данных (*extension data segment registers) ES, GS, FS*.

Указатель инструкций EIP содержит смещение адреса текущей инструкции, которое используется совместно с регистром *CS* для получения соответствующего виртуального адреса.

Регистр флагов EFLAGS содержит признаки, характеризующие результат выполнения операции, например, флаг знака, флаг нуля, флаг переполнения, флаг паритета, флаг переноса и некоторые другие. Кроме того, здесь хранятся некоторые признаки, устанавливаемые и анализируемые механизмом прерываний, в частности флаг разрешения аппаратных прерываний IF.

В процессоре *Pentium* имеется ***пять управляющих регистров – CRO, CR1, CR2, CR3 и CR4***, которые хранят признаки и данные, характеризующие общее состояния процессора.

Регистр *CR*0 содержит все основные признаки, существенно влияющие на работу процессора, такие как реальный/защищенный режим работы, включение/ выключение страничного механизма системы виртуальной памяти, а также признаки, влияющие на работу кэша и выполнение команд с плавающей точкой. Регистр *CR*1 в настоящее время не используется (зарезервирован). Регистры *CR*2 и *CR*3 предназначены для поддержки работы системы виртуальной памяти. В регистре *CR*4 хранятся признаки» разрешающие работу так называемых архитектурных расширений, например, возможности использования страниц размером 4 Мбайт и т. п.

Регистры системных адресов. Эти регистры еще называют регистрами управления памятью. Они предназначены для защиты программ и данных в мультизадачном режиме работы процессора. При работе в защищенном режиме процессора адресное пространство делится на:

- глобальное – общее для всех задач;
- локальное – отдельное для каждой задачи. Этим разделением и объясняется то, что в архитектуре процессора присутствуют следующие системные регистры:
- регистр таблицы глобальных дескрипторов *GDTR* (*global descriptor table register*) имеет размер 48 бит и содержит 32-битовый (биты 16–47) базовый адрес глобальной таблицы дескрипторов (*GDT*) и 16-битовое (биты 0–15) значение предельного размера, представляющее собой размер в байтах таблицы *GDT*;
- регистр таблицы локальных дескрипторов *LDTR* (*local descriptor table register*) имеет размер 16 бит и содержит так называемый селектор дескриптора локальной таблицы дескрипторов. Этот селектор является указателем в таблице *GDT*, который и описывает сегмент, в котором находится локальная таблица дескрипторов (*LDT*).
- регистр таблицы дескрипторов прерываний *IDTR* (*interrupt descriptor table register*) имеет размер 48 бит и содержит 32-битовый (биты 16–47) базовый адрес таблицы дескрипторов прерываний (*IDT*) и 16-битовое (биты 0–15) значение предела, представляющее собой размер в байтах таблицы *IDT*;
- 16-битовый регистр задачи *TR* (*task register*), подобно регистру *LDTR*, содержит селектор, то есть указатель на дескриптор в таблице *GDT*. Этот дескриптор описывает текущий сегмент состояния задачи (*TSS – task segment status*). Последний создается для каждой задачи в системе, имеет жестко регламентированную структуру и хранит контекст (текущее состояние) задачи. Основное назначение сегментов *TSS* – запоминать текущее состояние задачи в момент переключения на другую задачу.

Регистры отладки хранят значения точек останова, а ***регистры тестирования*** позволяют проверить корректность работы внутренних блоков процессора.

Средства поддержки сегментации памяти

Средства поддержки механизмов виртуальной памяти в процессоре *Pentium* позволяют отображать виртуальное адресное пространство на физическую память размером максимум в 4 Гбайт (этот максимум определяется использованием 32-разрядных адресов при работе с оперативной памятью).

Процессор может поддерживать как сегментную модель распределения памяти, так и сегментно-страничную. Средства сегментации образуют верхний уровень средств управления виртуальной памятью процессора *Pentium*, а средства страничной организации – нижний уровень. Это означает, что сегментные средства работают всегда, а средства страничной организации могут быть как включены, так и выключены путем установки однобитного признака *PE* (*Paging Enable*) в регистре *CR*0 процессора. В зависимости от того, включены ли средства страничной организации, изменяется смысл процедуры

преобразования адресов, которая выполняется средствами сегментации. Сначала рассмотрим случай работы средств сегментации при отключенном механизме управления страницами.

При работе процессора *Pentium* в сегментном режиме в распоряжении программиста имеется виртуальное адресное пространство, представляемое совокупностью сегментов.

Каждый сегмент виртуальной памяти процесса имеет описание, называемое дескриптором сегмента. Сегментный дескриптор – структура данных в глобальной таблице дескрипторов *GDT* или локальной таблице дескрипторов *LDT*, задающая для процессора информацию о размере и расположении сегмента, а также информацию о правах доступа и статусную информацию. Сегментные дескрипторы обычно создаются компиляторами, линкерами, загрузчиками или операционными системами. Дескриптор сегмента имеет размер 8 байт (рис. 2).

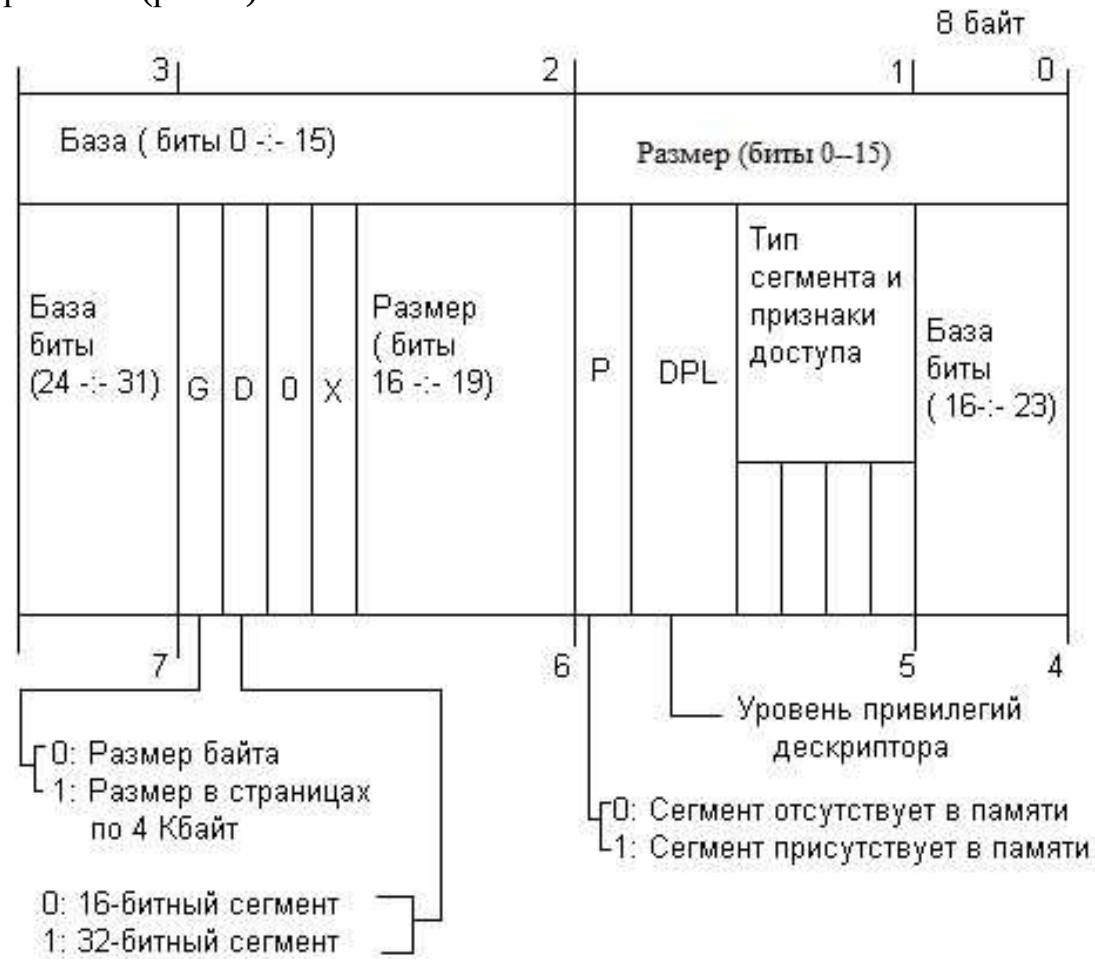


Рис. 2. Формат дескриптора сегмента данных или кода

Ниже перечислены основные поля дескриптора.

- База – базовый адрес сегмента (32 бита) – 2-й, 3-й, 4-й, 7-й байты.
- Размер – размер сегмента (24 бита) – 0-й, 1-й, 6-й байты.

- *G* (*Granularity*) – единица измерения размера сегмента, один бит. Если *G*=0, то размер задан в байтах и тогда сегмент не может быть больше 64 Кбайт, если *G* =1, то размер сегмента измеряется в страницах по 4 Кбайт.
- Байт доступа (5-й байт дескриптора) содержит информацию, которая используется для принятия решения о возможности или невозможности обращения к данному сегменту. Бит *P* (*Present*) определяет, находится ли соответствующий сегмент в данный момент в памяти (*P*=1) или он выгружен на диск *(P*=0). Поле *DPL* (*Descriptor Privilege Level*) содержит данные об уровне привилегий, необходимом для доступа к сегменту. Это 2-х битовое поле, в него записывают значения от 0 до 3. Остальные пять битов байта доступа зависят от типа сегмента и определяют способ, которым можно использовать данный сегмент (то есть читать, писать, выполнять).

Различаются три основных типа сегментов:

- сегмент данных;
- кодовый сегмент;
- системный сегмент.

Дескрипторы сегментов объединяются в таблицы. Процессор *Pentium* для управления памятью поддерживает два типа таблиц дескрипторов сегментов:

- глобальная таблица дескрипторов (*Global Descriptor Table, GDT*), которая предназначена для описания сегментов операционной системы и общих сегментов для всех прикладных процессов;
- локальная таблица дескрипторов (*Local Descriptor Table, LDT*), которая содержит дескрипторы сегментов отдельного пользовательского процесса.

Таблица *GDT* одна, а таблиц *LDT* столько, сколько в системе выполняется задач (процессов). При этом в каждый момент времени операционной системой и аппаратными средствами процессора используется только одна из таблиц *LDT*, а именно та, которая соответствует выполняемому в данный момент пользовательскому процессу. Таблица *GDT* описывает общую часть виртуального адресного пространства процессов, а *LDT* – индивидуальную часть для каждого процесса. Таблицы *GDT* и *LDT* размещены в оперативной памяти в виде отдельных сегментов.

Процесс обращается к физической памяти по виртуальному адресу, представляющему собой пару (селектор, смещение). Селектор однозначно определяет виртуальный сегмент, к которому относится искомый адрес, то есть он может интерпретироваться как номер сегмента, а смещение, как это и следует из его названия, фиксирует положение искомого адреса относительно начала сегмента. Смещение задается в машинной инструкции, а селектор помещается в один из сегментных регистров процессора. Под смещение отводится 32 бита, что обеспечивает максимальный размер сегмента 4 Гбайт (2^{32}).

Селектор извлекается из одного из шести 16-разрядных сегментных регистров процессора (*CS*, *SS*, *DS, ES, FS* или *GS*) в зависимости от типа команды и стадии ее выполнения – выборки кода команды или данных.

Виртуальное адресное пространство процесса складывается из всех сегментов, описанных в общей для всех процессов таблице *GDT*, и сегментов, описанных в его собственной таблице *LDT*. Разрядность поля индекса определяет максимальное число глобальных и локальных сегментов процесса – по 8 Кбайт (2^{13}) сегментов каждого типа, всего 16 Кбайт сегментов. С учетом максимального размера сегмента – 4 Гбайт – каждый процесс при чисто сегментной организации виртуальной памяти (без включения страничного механизма) может работать в виртуальном адресном пространстве в 64 Тбайт.

Теперь проследим, каким образом виртуальное пространство в 64 Тбайт отображается на физическое пространство размером в 4 Гбайт. Механизм отображения преобразовывает виртуальный адрес, который представлен селектором, находящимся в одном из сегментных регистров, и смещением, извлеченным из соответствующего поля машинной инструкции, в линейный физический адрес.

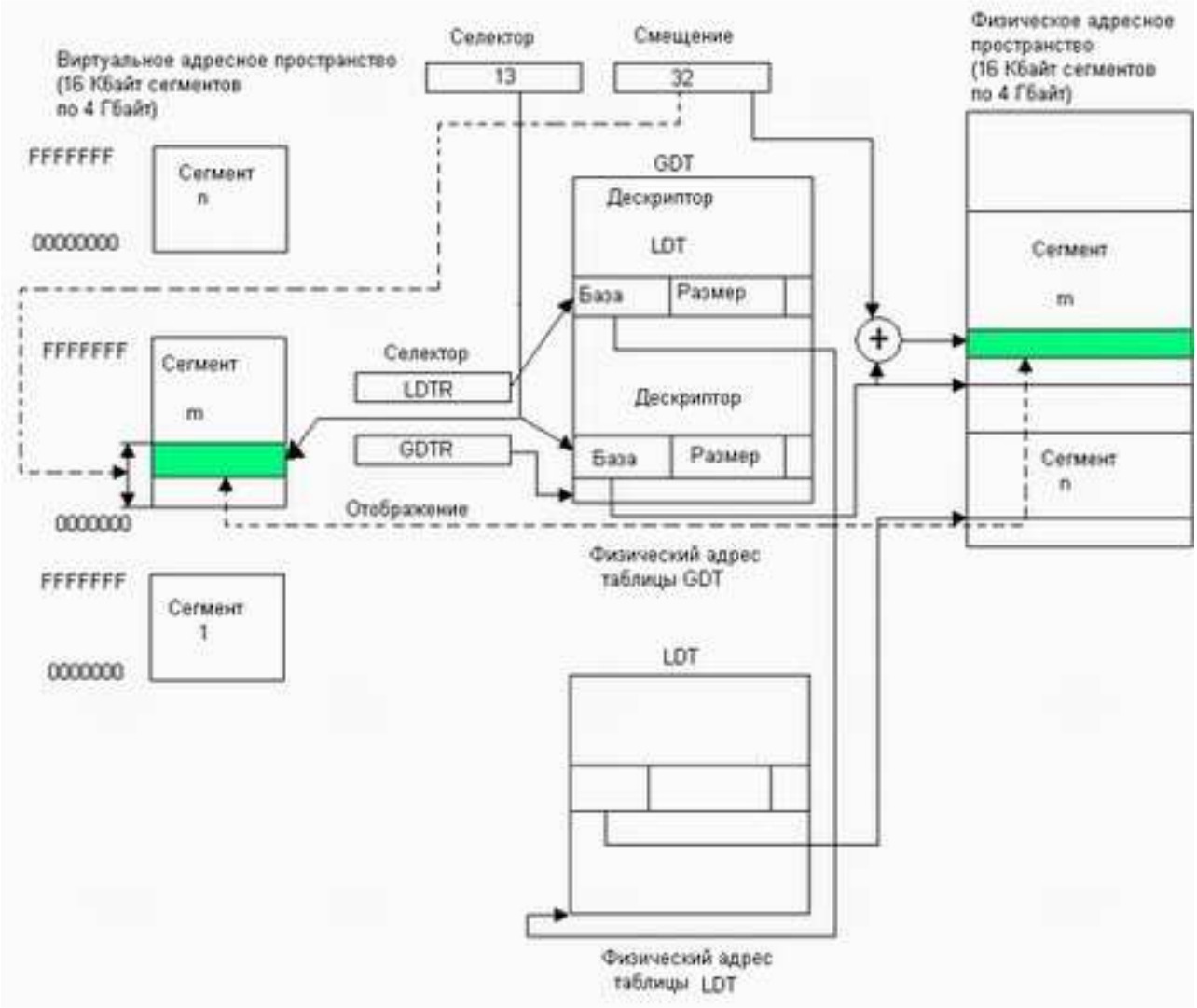


Рис. 3. Механизм преобразования виртуального адреса в физический при работе процессора в сегментном режиме

Рассмотрим случай, когда виртуальный адрес относится к одному из сегментов, дескрипторы которых содержатся в таблице *GDT* (рис. 3).

1. Значение селектора указывает механизму преобразования адресов, что виртуальный адрес относится к сегменту, описываемому в таблице *GDT*. Местонахождение таблицы *GDT* система определяет из регистра *GDTR*, в котором хранится полный 32-битный базовый физический адрес таблицы. Процессор складывает базовый адрес таблицы, взятый из регистра *GDTR*, со сдвинутым на 3 разряда влево (умножение на 8 в соответствии с числом байтов в одном дескрипторе сегмента) значением поля индекса из селектора. Результатом является физический адрес дескриптора сегмента, к которому относится заданный виртуальный адрес.

2. По вычисленному адресу процессор извлекает из памяти дескриптор нужного сегмента.

3. Выполняется проверка возможности выполнения заданной операции доступа по заданному виртуальному адресу:

- сначала процессор определяет правильность адреса, сравнивая смещение, заданное в виртуальном адресе, с размером сегмента, извлеченным из регистра *LDTR* (в случае выхода адреса за границы сегмента происходит прерывание);
- затем процессор проверяет права доступа задачи к данному сегменту памяти;
- далее проверяется наличие сегмента в физической памяти (если бит *P* дескриптора равен 0, то есть сегмент отсутствует в физической памяти, то происходит прерывание).

4. Если все три условия выполнены, то доступ по заданному виртуальному адресу разрешен. Выполняется преобразование виртуального адреса в физический путем сложения базового адреса сегмента, извлеченного из дескриптора, и смещения, заданного в инструкции. Выполняется заданная операция над элементом физической памяти по этому адресу.

Таким образом, для использования сегментного механизма процессора *Pentium* операционной системе необходимо сформировать таблицы *GDT* и *LDT*, загрузить их память (для начала достаточно загрузить только таблицу *GDT*), загрузить указатели на эти таблицы в регистры *GDTR* и *LDTR* и выключить страничную поддержку.

Процессор *Pentium* при работе в сегментном режиме предоставляет операционной системе следующие средства, направленные на обеспечение защиты процессов друг от друга.

- Процессор *Pentium* поддерживает для каждого процесса отдельную таблицу дескрипторов сегментов *LDT* и делает недоступным для процесса локальные сегменты других процессов, описанные в их таблицах *LDT*.
- Вместе с тем таблица *GDT*, в которой хранятся дескрипторы сегментов операционной системы, а также сегменты, используемые несколькими

процессами совместно, доступна всем процессам, а, следовательно, не защищена от их несанкционированного вмешательства. Для решения этой проблемы в процессоре *Pentium* предусмотрена поддержка системы безопасности на основе привилегий

- Еще одним механизмом защиты, поддерживаемым в процессоре *Pentium*, является ограничение на способ использования сегмента. В зависимости от того, к какому типу относится сегмент – сегмент данных, кодовый сегмент или системный сегмент, – некоторые действия по отношению к нему могут быть запрещены.

Сегментно-страничный механизм

Включение страничного механизма происходит, если в регистре управления *CR*0 самый старший бит *PG* установлен в единицу. При включенной системе управления страницами параллельно продолжает работать и описанный выше сегментный механизм, однако, как будет показано ниже, смысл его работы меняется.

Виртуальное адресное пространство процесса при сегментно-страничном режиме работы процессора ограничивается размером 4 Гбайт. В этом пространстве определены виртуальные сегменты процесса (рис. 4).

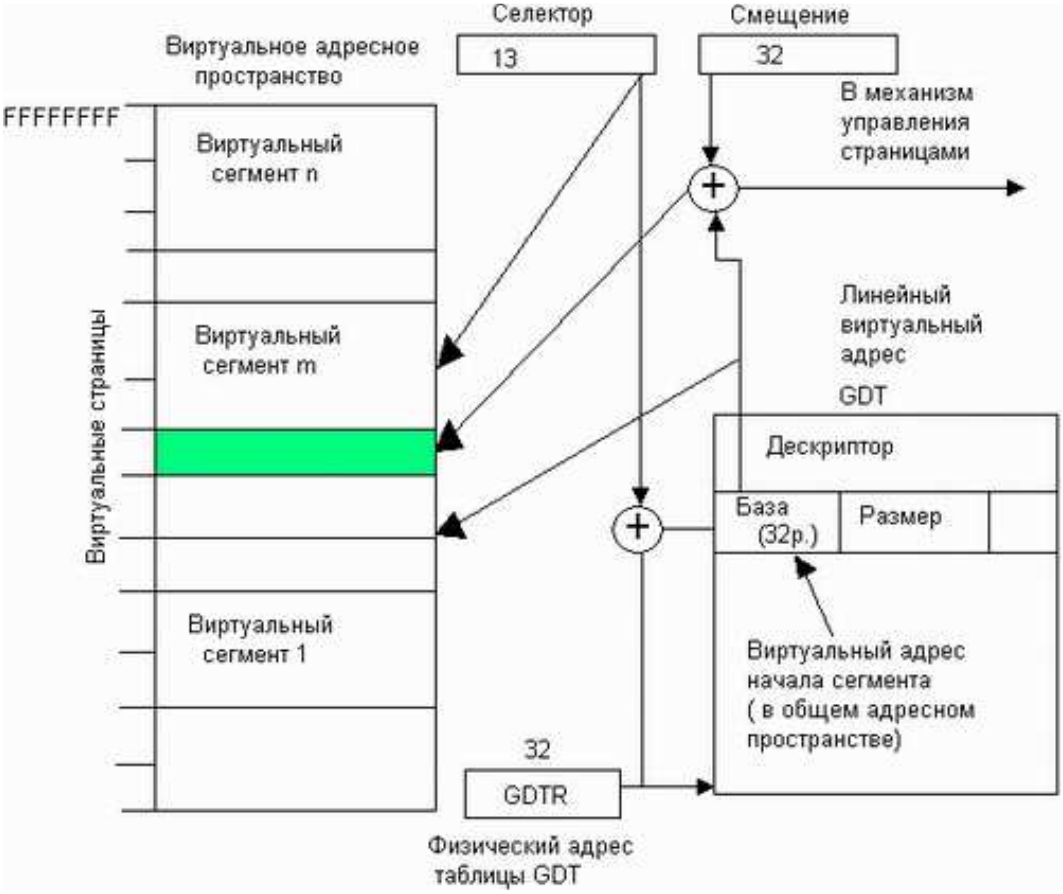


Рис. 4. Работа сегментного механизма в сегментно-страничном режиме

Для реализации механизма управления страницами как физическое, так и виртуальное адресные пространства разбиты на страницы размером 4 Кбайт. Всего в виртуальном адресном пространстве в сегментно-страничном режиме насчитывается 1 Мбайт (2^{20}) страниц. Несмотря на наличие нескольких виртуальных сегментов, все виртуальное адресное пространство задачи имеет общее разбиение на страницы, так что нумерация виртуальных страниц сквозная.

Виртуальный адрес по-прежнему представляет собой пару: селектор, который определяет номер виртуального сегмента, и смещение внутри этого сегмента. Преобразование виртуального адреса выполняется в два этапа: сначала работает сегментный механизм, а затем результат его работы поступает на вход страничного механизма, который и вычисляет искомый физический адрес.

Работа сегментного механизма в данном случае во многом повторяет его работу при отключенном страничном механизме. Однако имеется и принципиальное отличие, оно состоит в интерпретации содержимого поля базового адреса в дескрипторах сегментов.

Если раньше дескриптор сегмента содержал базовый адрес сегмента в физической памяти и при сложении этого адреса со смещением из виртуального адреса получался физический адрес, то теперь дескриптор содержит базовый адрес сегмента в виртуальном адресном пространстве, и в результате его сложения со смещением получается линейный виртуальный адрес.

Результирующий линейный 32-разрядный виртуальный адрес передается страничному механизму для его преобразования в физический адрес (рис. 5).

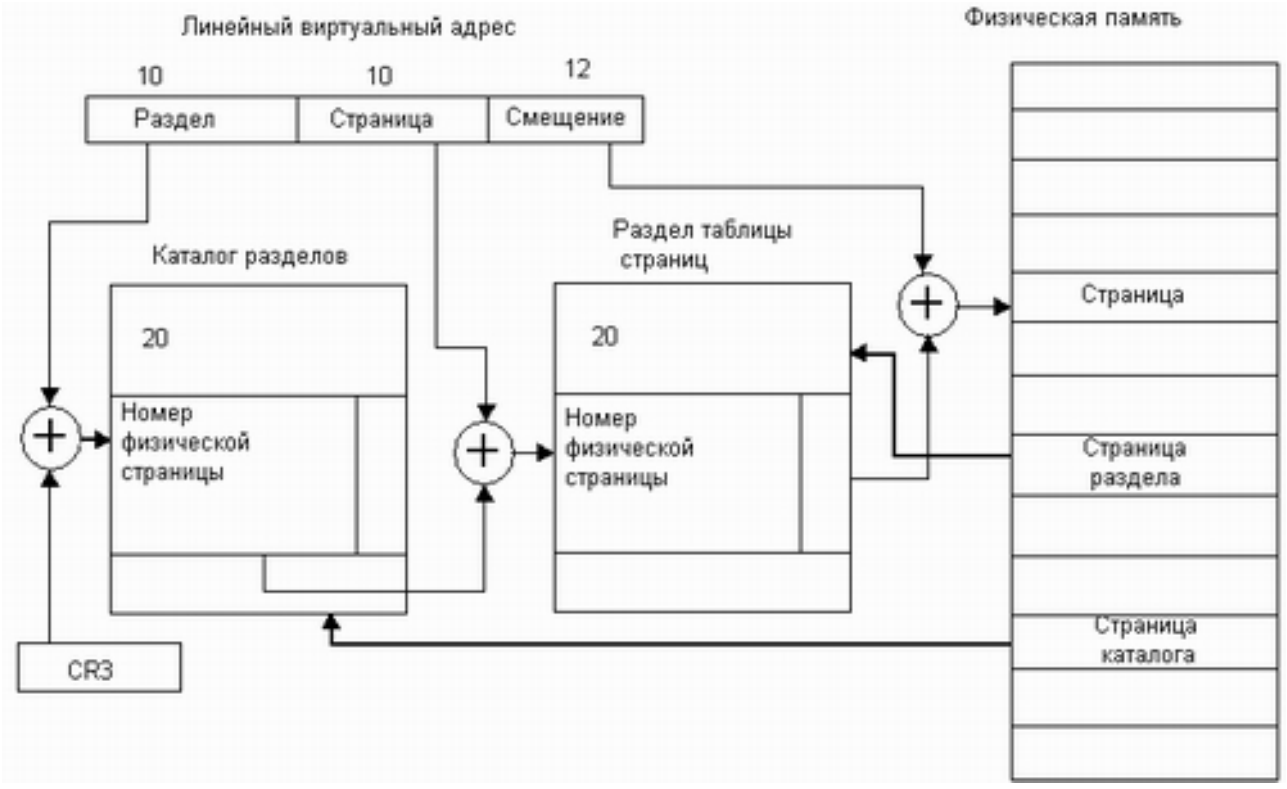


Рис. 5. Преобразование линейного виртуального адреса в физический адрес

Поле номера виртуальной страницы (старшие 20 разрядов) делится на две равные части по 10 разрядов – поле номера раздела и поле номера страницы в разделе. На основании заданного в регистре *CR*3 номера физической страницы, хранящей таблицу разделов, и смещения в этой странице, задаваемого полем номера раздела, процессор находит дескриптор виртуальной страницы раздела. В соответствии с атрибутами этого дескриптора определяются права доступа к странице, а также наличие ее в физической памяти. Если страницы нет в оперативной памяти, то происходит прерывание, в результате которого операционная система должна выполнить загрузку требуемой страницы в память. После того как страница (содержащая нужный раздел) загружена, из нее извлекается дескриптор страницы данных, номер которой указан в линейном виртуальном адресе. И наконец, на основании базового адреса страницы, полученного из дескриптора, и смещения, заданного в линейном виртуальном адресе, вычисляется искомый физический адрес.

Таким образом, при доступе к странице в процессоре используется двухуровневая схема адресации страниц, которая хотя и замедляет преобразование, но позволяет использовать страничный механизм для таблицы страниц, что существенно уменьшает объем физической памяти, требуемой для ее хранения.

Механизм прерываний

Процессор *Pentium* поддерживает 256 прерываний. Каждое из них имеет свой номер и свою процедуру обработки прерывания (*ISR- Interrupt Serves Routine*).

Прерывания, которые обрабатывает процессор Pentium делятся на следующие классы:

- *аппаратные (внешние)* – источником таких прерываний является сигнал на входе процессора;
- *исключения* – внутренние прерывания процессора;
- *программные прерывания*, происходящие по команде *INT*.

Аппаратные прерывания бывают маскируемыми и немаскируемыми. Маскируемые прерывания вызываются сигналом *INTR* на одном из входов микросхемы процессора. Маскируемость прерываний управляется флагом разрешения прерываний *IF*, находящимся в регистре *EFLAGS* процессора. При *IF*=1 маскируемые прерывания разрешены, при *IF*=0 – запрещены. Для маскируемых прерываний отведены *ISR* с номерами 32-255.

Имеется 16 линий запросов на прерывания (*IRQ – **I**nterrupt **R**e**Q**uest*), реализуемых с помощью двух контроллеров прерываний *i*8259*A*, соединенных каскадным образом.

Немаскируемое прерывание происходит при появлении сигнала *NMI* (*Non Maskable Interrupt*) на входе процессора. Этот сигнал всегда прерывает работу процессора, вне зависимости от значения флага *IF*. Управление всегда передается **ISR** с номером 2. Эти прерывания предназначаются для реакции на «сверхважные» для компьютерной системы события, например, сбой по питанию.

Исключения (exeptions) делятся на отказы (faults), ловушки (traps) и аварийные завершения (aborts).

Отказы соответствуют некорректным ситуациям, которые выявляются до выполнения инструкции, например, при обращении к странице, которой нет в оперативной памяти.

Ловушки обрабатываются процессором после выполнения инструкции, например, при возникновении переполнения.

Аварийные завершения соответствуют ситуациям, когда невозможно точно определить команду, вызвавшую прерывание. Это происходит при сбоях в работе аппаратуры компьютера.

Для обработки исключений в таблице прерываний отводятся номера 0-31.

Программные прерывания происходят при выполнении инструкции *INT* с однобайтовым аргументом, в котором указывается вектор прерывания (фактически указывается номер процедуры обработки прерывания). Программные прерывания обрабатываются как ловушки после выполнения инструкции *INT*, а возврат происходит в следующую инструкцию. Программное прерывание может вызвать любую из 256 *ISR*.

При одновременном возникновении запросов прерываний различных типов процессор *Pentium* разрешает коллизию с помощью приоритетов. Немаскируемые прерывания имеют более высокий приоритет, чем маскируемые. Приоритетность внутри маскируемых прерываний устанавливается не процессором, а контроллером прерываний (процессор не может этого сделать, так как для него все маскируемые запросы представлены одним сигналом *INTR*). Проверка некорректных ситуаций, порождающих исключения (в том числе и при выполнении одной команды), выполняется в процессоре в соответствии с определенной последовательностью.

В реальном режиме работы система прерываний процессора *Pentium* работает так же, как и для 16-разрядного микропроцессора *i*8086/8088. Адрес точки входа в *ISR* называется вектором *прерываний* и хранится в специальной таблице, называемой *таблицей векторов прерывания (*ТВП*)*. Код *ISR* может располагаться в любом месте памяти. Поэтому вектор прерывания занимает 4 байта:

- 2 байта отводится на значение сегмента адреса – значение регистра *CS*;
- 2 байта – на значение смещения, устанавливаемое в регистре *IP*.

Вся ТВП занимает 256*4 = 1024 байта и хранится в фиксированном месте физической памяти, начиная с адреса 0000 по адрес 003*FF*.

При возникновении прерывания процессор помещает в стек 6 байт:

- текущее значение регистра *CS*;
- текущее значение регистра *IP*;
- 2 байта флагов процессора;
- в *CS* и *IP* устанавливаются значения из ТВП, задающие начальный адрес *ISR*.

Прерыванию «0» соответствует вектор прерывания по адресу 0000:0000. Прерыванию «1» соответствует вектор прерывания по адресу 0000:0004*h*.Прерыванию «2» соответствует вектор прерывания по адресу 0000:00008*h* и т.д.

В защищенном режиме таблица прерываний носит название *IDT* (*Interrupt Descriptor Table*) и может располагаться в любом месте физической памяти. Ее начало (32-разрядный физический адрес) и размер (16 бит) можно найти в регистре системных адресов *IDTR*. Каждый из 256 элементов таблицы прерываний представляет не 4-х байтный вектор прерывания, как в реальном режиме, а 8-байтный дескриптор определенного типа – дескрипторы шлюзов прерывания, шлюзов ловушек и шлюзов задач.

Шлюзы задач используются для переключения с задачи на задачу. Шлюзы прерываний и ловушек специально вводятся для вызова процедур обработки прерываний. Если для вызова *ISR* используется шлюз задач, то происходит смена процесса, а по завершении обработки – возврат к прерванному процессу.

Шлюзы прерываний и ловушек не вызывают смены контекста задачи, поэтому *ISR* вызываются быстрее. При вызове *ISR* через шлюз прерываний. запрещаются вложенные прерывания.

Итак, процессор *Pentium* предоставляет операционной системе широкий диапазон возможностей для организации обработки прерываний различного типа.

Файловая система

Определение файловой системы звучит так: *это совокупность алгоритмов и стандартов, направленных на обеспечение доступа пользователя к данным, хранящимся на компьютере*. Специалисты относят файловую систему к разным категориям ПО, одни считают ее частью ОС, другие выделяют в качестве независимого компонента. Использование конкретных вариантов файловых систем обеспечивает определение размера и другие особенности в наименовании файлов и их групп (каталогов).

Файловая система обеспечивает взаимосвязь между информационным носителем (оптический или жесткий диск, флеш-накопители, карта памяти и др.) и *API* (программным интерфейсом, выполняющим служебные функции по обеспечению работы с различными приложениями, включая доступ к файлам).

Обращение к файлу – обязательный элемент работы используемых программ, функции которых завязаны на конкретном имени, размере и атрибутах файла. Приложение ориентируется на конкретный путь независимо от того, на какой тип носителя он ведет и как структурирована информация в пункте назначения. Операционная система воспринимает содержимое носителя, как набор кластеров приемлемого для восприятия размера (обычно 512 байт или больше). Пользователь видит документы и иерархическую структуру папок, так как файлы и каталоги формируются на базе кластеров посредством драйверов файловой системы, которые обеспечивают различение свободных и занятых участков, выявление неисправных кластеров.

Разработаны различные виды файловых систем. Объединение файлов базируется на иерархии посредством каталогов разного уровня, часто используют термины «подкаталоги», «папки», «директории». Каталоги – самые главные папки на диске или другом носителе, далее идут подкаталоги. Простой вариант хранения файлов – один каталог, соответствующий диску. Это одноуровневая файловая система, которая используется на дисках небольшого размера. Поиск файлов здесь проводится только по названию. Такую систему не используют в современных электронных устройствах, Затем была разработана иерархия с каталогами разного уровня, вложенными друг в друга. Файловая система *Windows* отличается присутствием нескольких отдельных деревьев. Другой вариант – объединение в одно дерево характерно для дисков и систем наподобие *UNIX*. Здесь для получения доступа к файлам нужно провести монтирование диска с помощью команды «*mount*». Это позволяет операционной системе увидеть и отобразить компоненты каталога «*/mnt/cdrom*» (его называют точкой монтирования).

Функционирование файловой системы включено в механизм обращения к аппаратным ресурсам компьютера, то есть магнитным или лазерным носителям (жесткие и оптические диски, флеш-накопители). Файлом на языке *IT*-экспертов называется область информации фиксированной величины, которая выражается в байтах. Конечные единицы объединяются в блоки с конкретным «адресом» доступа, координаты которого определяются файловой системой, доводятся до

сведения ОС и отображаются для пользователя в доступном формате. С помощью служебных программ данные считываются, модифицируются, формируются новые. Файловые системы используют разный алгоритм работы в отношении координат файлов, который зависит от:

- типа компьютера;
- категории операционной системы;
- особенности хранения и пр.

Форматирование – адаптация носителя к конкретной файловой системе. Кластеры устройства формируются определенным образом для обеспечения корректной записи файлов с последующим чтением на базе стандартов конкретной системы управления. Смена файловой системы в основном происходит посредством переформатирования носителя, что неизбежно приводит к стиранию файлов, однако есть специализированные ПО, позволяющие провести модификацию устройства без затрагивания на нем информации. Создание и размещение файлов на компьютере завязано на организации пространства посредством файловой системы, что обеспечивает комфортность поиска и обращения с нужной информации, пользователь не ощущает сложности алгоритмов доступа к данным.

Физическая организация жесткого диска

Основным типом устройств, которые используются в современных вычислительных системах для хранения файлов, являются дисковые накопители. Эти устройства предназначены для считывания и записи данных на жесткие и гибкие магнитные диски. Жесткий диск состоит из одной или нескольких стеклянных, или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом. Таким образом, диск в общем случае состоит из пакета пластин (рис. 1).

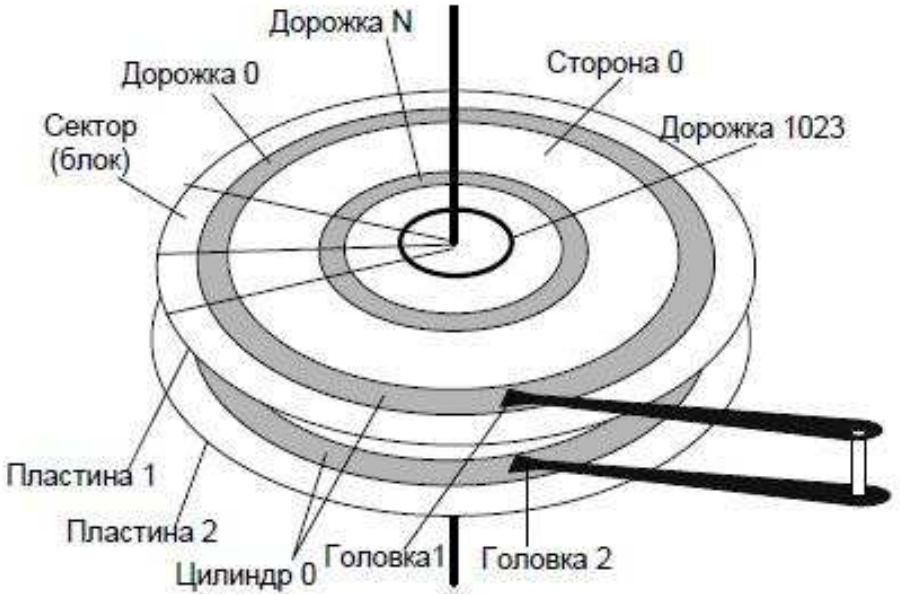


Рис. 1. Схема устройства жесткого диска

На каждой стороне каждой пластины размечены тонкие концентрические кольца – дорожки (*traks*), на которых хранятся данные. Количество дорожек зависит от типа диска. Нумерация дорожек начинается с 0 от внешнего края к центру диска. Когда диск вращается, элемент, называемый головкой, считывает двоичные данные с магнитной дорожки или записывает их на нее.

Головка может позиционироваться над заданной дорожкой. Головки перемещаются над поверхностью диска дискретными шагами, каждый шаг соответствует сдвигу на одну дорожку. Запись на диск осуществляется благодаря способности головки изменять магнитные свойства дорожки. В некоторых дисках вдоль каждой поверхности перемещается одна головка, а в других – имеется по головке на каждую дорожку. В первом случае для поиска информации головка должна перемещаться по радиусу диска. Обычно все головки закреплены на едином перемещающем механизме и двигаются синхронно. Поэтому, когда головка фиксируется на заданной дорожке одной поверхности, все остальные головки останавливаются над дорожками с такими же номерами. В тех же случаях, когда на каждой дорожке имеется отдельная головка, никакого перемещения головок с одной дорожки на другую не требуется, за счет этого экономится время, затрачиваемое на поиск данных.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин пакета называется цилиндром (*cylinder*). Современные диски имеют по несколько десятков тысяч таких цилиндров. Каждая дорожка разбивается на фрагменты, называемые секторами (*sectors*) или блоками (*blocks*). На первых жестких дисках все дорожки имели равное число секторов, в которые можно максимально записать одно и то же число байт. Сектор имел фиксированный для конкретной системы размер, выражающийся степенью двойки. Учитывая, что дорожки разного радиуса имели одинаковое число секторов, плотность записи

становилась тем выше, чем ближе дорожка к центру. Однако, современные жесткие диски имеют постоянную плотность записи, поэтому на разных дорожках располагается различное количество секторов.

Сектор – наименьшая адресуемая единица обмена данными дискового устройства с оперативной памятью. Физически адрес сектора на диске определяется триадой [**c-h-s**], где **c** – номер цилиндра (**c**ylinder), **h** – номер рабочей поверхности диска (магнитной головки, **h**ead), а **s** – номер сектора на дорожке (**s**ector). Для того чтобы контроллер мог найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора **–** [**c-h-s**].

Операционная система при работе с диском использует, как правило, собственную единицу дискового пространства, называемую кластером. При создании файла место на диске ему выделяется кластерами. Например, если файл имеет размер 2560 байт, а размер кластера в файловой системе определен в 1024 байта, то файлу будет выделено на диске 3 кластера.

Дорожки и секторы создаются в результате выполнения процедуры физического, или низкоуровневого форматирования диска, предшествующей использованию диска. Для определения границ блоков на диск записывается идентификационная информация. Низкоуровневый формат диска не зависит от типа ОС, которая этот диск будет использовать.

Разметку диска под конкретный тип файловой системы выполняют процедуры высокоуровневого, или логического, форматирования. При высокоуровневом форматировании определяется размер кластера и на диск записывается информация, необходимая для работы файловой системы, в том числе информация о доступном и неиспользуемом пространстве, о границах областей, отведенных под файлы и каталоги, информация о поврежденных областях.

Жесткий диск может быть разбит на несколько разделов (*partition*), которые могут использоваться либо одной ОС, либо различными ОС. При этом на каждом разделе может быть организована своя файловая система.

Разделы диска могут быть двух видов – первичный (*primary*) и расширенный (*extended*). Максимальное число *primary*-разделов равно четырем. При этом на диске обязательно должен быть один *primary*-раздел. Если *primary*-разделов несколько, то только один из них может быть активным. Именно загрузчику, расположенному в активном разделе, передается управление после тестирования *BIOS* и загрузке операционной системы. Остальным *primary*-разделам присваивается атрибут «*hidden*» - скрытый.

В первом физическом секторе жесткого диска располагается головная запись загрузки и таблица разделов (табл. 1).

Главная запись загрузки (*master boot record, MBR*) – первая часть данных на жестком диске. Она зарезервирована для программы начальной загрузки *BIOS* (*ROM Bootstrap routine*), которая при загрузке с жесткого диска считывает и загружает в память первый физический сектор на активном разделе диска, называемый загрузочным сектором (*Boot Sector*). Программа,

расположенная в *MBR*, носит название внесистемного загрузчика (*Non-System Bootstrap, NSB*).

Каждая запись в таблице разделов (*partition table*) содержит начальную позицию и размер раздела на жестком диске, а также информацию о том, первый сектор какого раздела содержит загрузочный сектор.

Таблица 1

Головная запись загрузки и таблица разделов

Размер (байт)	Описание
446	Загрузочная запись (MBR)
16	Запись 1 раздела
16	Запись 2 раздела
16	Запись 3 раздела
16	Запись 4 раздела
2	Сигнатура 055AAh

Можно сказать, что таблица разделов – одна из наиболее важных структур данных на жестком диске. Если эта таблица повреждена, то не только не будет загружаться ни одна из установленных на компьютере ОС, но станут недоступными данные, расположенные в диске, особенно если жесткий диск был разбит на несколько разделов.

Последние два байта таблицы разделов имеют значение 055*AAh*, то есть чередующиеся значения 0 и 1 в двоичном представлении данных. Эта сигнатура выбрана для того, чтобы проверить работоспособность всех линий передачи данных. Значение 055*AAh*, присвоенное последним двум байтам, имеется во всех загрузочных секторах.

Многие ОС позволяют создавать, так называемый, расширенный (*extended*) раздел, который по аналогии с разделами может разбиваться на несколько логических дисков (*logical disks*). Расширенный раздел содержит вторичную запись *MBR* (*Secondary MBR, SMBR*), в состав которой вместо таблицы разделов входит аналогичная ей таблица логических дисков (*Logical Disks Table, LDT*). Таблица логических дисков описывает размещение и характеристики раздела, содержащего единственный логический диск, а также может специфицировать следующую запись

Логическое устройство может быть создано и на базе нескольких разделов, причем эти разделы не обязательно должны принадлежать одному физическому устройству. Объединение нескольких разделов в единое логическое устройство может выполняться разными способами и преследовать разные цели, основные из которых: увеличение общего объема логического раздела, повышение производительности и отказоустойчивости.

На разных логических устройствах одного и того же физического диска могут располагаться файловые системы разного типа. На рис. 2 показан пример

диска, разбитого на три раздела, в которых установлены две файловых системы *NTFS* (разделы *C* и *E*) и одна файловая система *FAT* (раздел *D*).

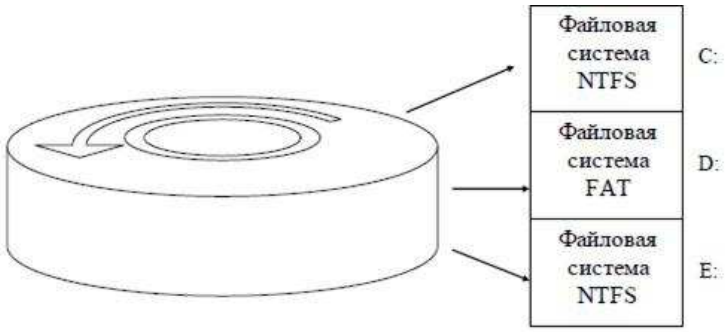


Рис. 2. Диск с тремя разделами, на которых установлены разные файловые системы

Цели и задачи файловой системы

Файл – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Файлы хранятся в энергонезависимой памяти, обычно – на магнитных дисках.

Основными целями использования файла являются:

1. Долговременное и надежное хранение информации. Долговременность достигается за счет использования запоминающих устройств, не зависящих от питания, а высокая надежность определяется средствами защиты доступа к файлам и общей организацией программного кода ОС, при которой сбои аппаратуры чаще всего не разрушают информацию, хранящуюся в файлах.

2. Совместное использование информации. Файлы обеспечивают естественный и легкий способ разделения информации между приложениями и пользователями за счет наличия понятного человеку символьного имени и постоянства хранимой информации и расположения файла. Пользователь должен иметь удобные средства работы с файлами, включая каталоги-справочники, объединяющие файлы в группы, средства поиска файлов по признакам, набор команд для создания, модификации и удаления файлов.

Файловая система, являющаяся неотъемлемой частью любой современной ОС, включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами (каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске);
- комплекс системных программных средств, реализующих различные операции над файлами (создание, уничтожение, чтение, запись, именование и поиск файлов).

Файловая система позволяет программам обходиться набором относительно простых операций для выполнения действий над некоторым

абстрактным объектом, представляющим файл. При этом программистам не нужно иметь дело с деталями действительного расположения данных на диске, буферизацией данных и другими низкоуровневыми проблемами передачи данных с долговременного запоминающего устройства – все эти функции файловая система берет на себя. Файловая система распределяет дисковую память, поддерживает именование файлов, отображает имена файлов в соответствующие адреса во внешней памяти, обеспечивает доступ к данным, поддерживает разделение, защиту и восстановление файлов.

Основными задачами файловой системы многопользовательской многозадачной ОС являются:

- идентификация файлов – связывание имени файла с выделенным ему пространством внешней памяти;
- распределение внешней памяти между файлами – для работы с конкретным файлом не требуется иметь информацию о местоположении этого файла на внешнем носителе информации (сторона магнитного диска, цилиндр, сектор);
- обеспечение надежности и отказоустойчивости;
- обеспечение защиты от несанкционированного доступа;
- обеспечение совместного доступа к файлам (пользователь не должен прилагать специальных усилий по обеспечению синхронизации доступа);
- обеспечение высокой производительности.

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых, как правило, входят:

- *обычные файлы* (содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ);
- *файлы-каталоги* (содержат системную справочную информацию о наборе файлов, сгруппированных пользователями по какому-либо признаку);
- *специальные файлы* (фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам);
- *отображаемые в память файлы* (мощная возможность ОС, позволяющая приложениям осуществлять доступ к файлам на диске тем же самым способом, каким осуществляется доступ к динамической памяти, то есть через указатели);
- *именованные конвейеры* (одно из средств межпроцессного взаимодействия);
- *другие*.

Кроме имени ОС часто связывают с каждым файлом и другую информацию, например, дату модификации, размер и т.д. Эти характеристики файлов называют *атрибутами*. В разных файловых системах могут использоваться в качестве атрибутов разные характеристики.

Очевидно, что конкретные файловые системы могут обладать различными особенностями – иметь различные ограничения на имена файлов (количество и

допустимые типы символов) и значения их расширения (позволяют определять тип файла или просто являются частью имени файла), поддерживать различный набор атрибутов и их предназначение и т.п.

Учитывая это, рассмотрим подробнее особенности функционирования конкретных файловых систем – как традиционных (*FAT* 32), так и новых, широко используемых в настоящее время (*NTFS*).

Файловая система *FAT*

Файловая система *FAT* (*File Allocation Table*) была разработана Биллом Гейтсом и Марком Мак Дональдом в 1977 году и стала основой для ОС *MS-DOS* 1.0, выпущенной в августе 1981 года.

Файловая система *FAT* была предназначена для работы с гибкими дисками размером менее 1 Мбайт, и вначале не предусматривала поддержки жестких дисков. В настоящее время *FAT* поддерживает файлы и разделы размером до 2 Гбайт.

В *FAT* применяются следующие соглашения по именам файлов:

- имя должно начинаться с буквы или цифры и может содержать любой символ *ASCII*, за исключением пробела и символов «" / \ [ ] : ; | = , ^ * ? »;
- длина имени не превышает 8 символов, за ним следует точка и необязательное расширение длиной до 3 символов;
- регистр символов в именах файлов не различается и не сохраняется.

***F**ile **A**llocation **T**able* (таблица размещения файлов) – линейная табличная структура со сведениями о файлах. В файловой системе *FAT* логическое дисковое пространство любого логического диска делится на две области: системную область и область данных. Системная область создается и инициализируется при форматировании и состоит из следующих компонент, расположенных друг за другом (рис.3):

1. Загрузочная запись (boot record – *BR*);
2. Зарезервированные сектора (reserved sector – *RS*);
3. Таблицы размещения файлов (*FAT*);
4. Корневой каталог (root directory – *Rdir*).

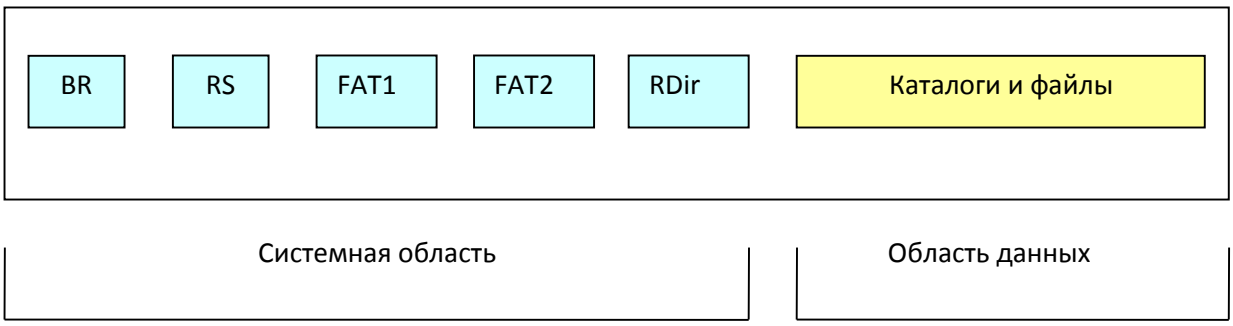


Рис. 3. Структура логического раздела в файловой системе *FAT*

FAT представляет карту области данных, в которой описывается состояние каждого участка области данных. Область данных разбивают на *кластеры*. Кластер – это один или несколько смежных секторов в логическом адресном пространстве области данных. В таблице *FAT* кластеры, принадлежащие одному файлу (или некорневому каталогу), связываются в цепочки. Для указания номера кластера в *FAT*16 используется 16-битовое слово, следовательно, можно иметь до 2^{16} = 65536 кластеров (с номерами от 0 до 65535). В *FAT* 32 в 32-битовом слове фактически учитывается только 28 разрядов, поэтому количество кластеров не может превышать 2^{28} = 268 435 456.

Кластер – это минимальная адресуемая единица дисковой памяти, выделяемая файлу (или некорневому каталогу). Файл или каталог занимает целое число кластеров. Последний кластер при этом может быть задействован не полностью, что приведет к заметной потере дискового пространства при большом размере кластера. На дискетах кластер занимает1 сектор, а на жестких дисках – в зависимости от объема раздела от 2 до 32 секторов. Слишком большой размер кластера ведет к неэффективному использованию дискового пространства, особенно в случае большого количества маленьких файлов.

Первый допустимый номер кластера всегда начинается с номера 2. Номера кластеров всегда соответствуют элементам таблицы размещения файлов.

Наглядно идею файловой системы *FAT* можно проиллюстрировать с помощью рис.4.

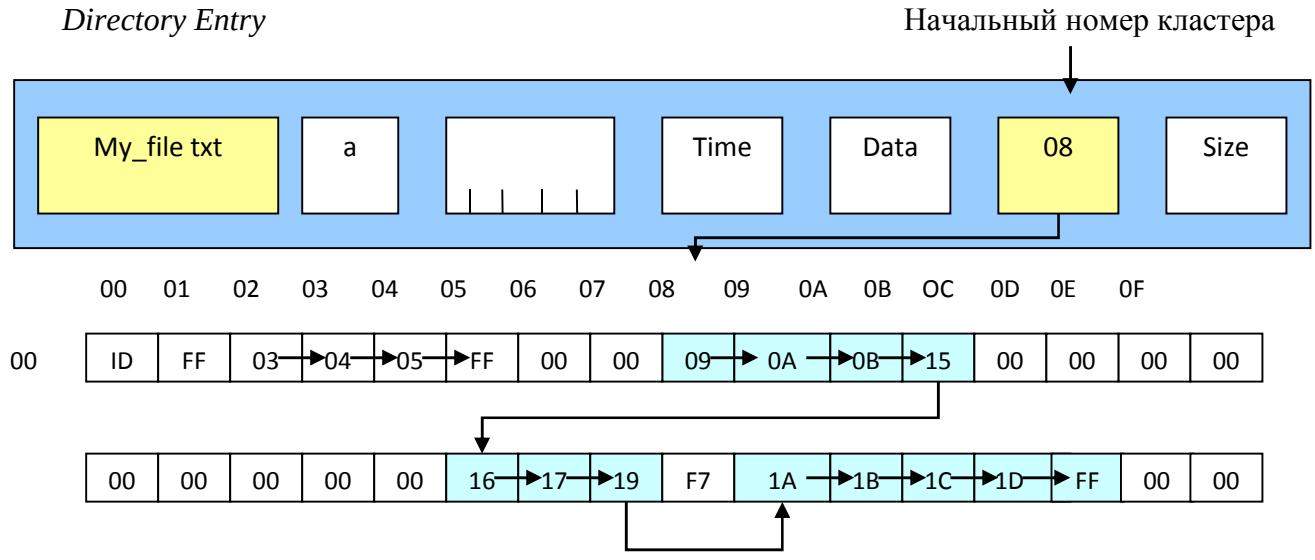


Рис. 4. Основная концепция *FAT*

Начальный номер кластера записывается в элемент каталога (*Directory Entry*). Таблица *FAT* имеет столько элементов, сколько имеется кластеров на диске. В элемент таблицы *FAT* с номером, соответствующим номеру кластера цепочки, записывается номер следующего кластера цепочки. При форматировании диска в элементы *FAT*, соответствующие дефектным

кластерам, записывается код *F*7. Свободные кластеры помечаются кодом 00. В элемент *FAT*, соответствующий последнему кластеру цепочки, записывается код *FF*.

Из рис. 4 видно, что файл с именем *My_file.txt* размещается, начиная с восьмого кластера. Цепочка кластеров для этого файла имеет вид:

8, 9, 0*A*, 0*B*, 15, 16, 17, 19, 1*A*, 1*B*, 1*C*, 1*D*

При выделении нового кластера для записи файла берется первый свободный кластер. Т.к. файлы в процессе работы изменяются, то это правило приводит к *фрагментации* файлов, т.е. данные одного файла могут располагаться не в смежных кластерах, образуя сложные цепочки. Это приводит к замедлению работы с файлами.

В связи с чрезвычайной важностью *FAT* она обычно хранится в двух идентичных экземплярах, второй из которых непосредственно следует за первым.

Для каждого файла и каталога в файловой системе хранится следующая информация: имя файла или каталога, атрибуты файла, резервное поле, время создания, дата создания, дата последнего доступа, зарезервировано, время последней модификации, дата последней модификации, номер начального кластера в *FAT*, размер файла.

Структура системы файлов является иерархической. Элементом каталога может быть такой файл, который сам, в свою очередь, является каталогом.

Файловая система *NTFS*

Файловая система *NTFS* (*New Technology File System*) разрабатывалась *Microsoft* в начале 1990-х гг. как основная файловая система для серверных версий операционных систем W*indows*. *NTFS* была представлена в 1993 году в операционной системе *Windows NT* 3.1.

В название файловой системы *NTFS* входят слова «новая технология». *NTFS* содержит ряд значительных усовершенствований и изменений, существенно отличающих ее от других файловых систем. С точки зрения пользователей, файлы по-прежнему хранятся в каталогах (часто называемых «папками»). Однако в *NTFS* в отличие от *FAT* работа на дисках большого объема происходит намного эффективнее; имеются средства для ограничения в доступе к файлам и каталогам, введены механизмы, существенно повышающие надежность файловой системы, сняты многие ограничения на максимальное количество дисковых секторов и/или кластеров.

Основные возможности файловой системы NTFS:

- *Надежность*. Одним из способов увеличения надежности является введение механизма транзакций, при котором осуществляется журналирование файловых операций.

• *Расширенная функциональность*. *NTFS* проектировалась с учетом возможного расширения. В ней были воплощены многие дополнительные возможности – усовершенствованная отказоустойчивость, эмуляция других файловых систем, мощная модель безопасности, параллельная обработка потоков данных и создание файловых атрибутов, определяемых пользователем.

• *Поддержка POSIX (*переносимый интерфейс операционных систем*)*. Поскольку правительство США требовало, чтобы все закупаемые им системы хотя бы в минимальной степени соответствовали стандарту *POSIX*, такая возможность была предусмотрена и в *NTFS*.

• *Гибкость*. Модель распределения дискового пространства в *NTFS* отличается чрезвычайной гибкостью. Размер кластера может изменяться от 512 байт до 64 Кбайт, стандартом же считается кластер размером 2 или 4 Кбайт. Он представляет собой число, кратное внутреннему кванту распределения дискового пространства. *NTFS* также поддерживает длинные имена файлов, набор символов *Unicode*. В *NTFS* используются 64 разрядные идентификаторы кластеров, поэтому теоретически том *NTFS* может содержать 2^{64} кластеров ($16*2^{60}$= 16 ЭБ – 16000 млрд. гигабайт). Однако текущие реализации в *Windows* поддерживают только 32 разрядную адресацию кластеров, что при размере кластера максимум 64 КБ (2^{16} байт) позволяет *NTFS* тому достигать размера до 256 ТБ = $2^{32} * 2^{16}$ байт = 2^{48} байт = $2^{8} * 2^{40}$ байт =256 ТБ.

Количество файлов в корневом и некорневом каталогах не ограничено. Поскольку в основу структуры каталогов *NTFS* заложена эффективная структура данных, называемая «бинарным деревом», время поиска файлов в *NTFS* (в отличие от систем на базе *FAT*) не связано линейной зависимостью с их количеством.

Структура тома с файловой системой NTFS

Одним из основных понятий, используемых при работе с *NTFS*, является понятие тома (*volume*). Возможно также создание отказоустойчивого тома, занимающего несколько разделов, то есть использование *RAID*-технологии (*избыточный массив независимых дисков*— технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности). Как и многие другие системы, *NTFS* делит все полезное дисковое пространство тома на кластеры – блоки данных, адресуемые как единицы данных.

Все дисковое пространство в *NTFS* делится на две неравные части (рис. 5). Первые 12% диска отводятся под так называемую *MFT*-зону – пространство, которое может занимать, увеличиваясь в размере, главный служебный метафайл *MFT*.

Рис. 5. Структура *NTFS* тома

Запись каких-либо данных в эту область невозможна. *MFT*-зона всегда держится пустой – это делается для того, чтобы самый главный, служебный файл *MFT* по возможности не фрагментировался при своем росте. Остальные 88% тома представляют собой обычное пространство для хранения файлов.

MFT (*master file table* – общая таблица файлов) состоит из записей фиксированного размера. Размер записи *MFT* (минимум 1 Кб и максимум 4 Кб) определяется во время форматирования тома. *MFT* представляет собой централизованный каталог всех остальных файлов диска, в том числе и себя самого. Каждая запись соответствует какому-либо файлу. Первые 16 файлов носят служебный характер и недоступны операционной системе – они называются метафайлами, причем самый первый метафайл – сам *MFT*. Эти первые 16 элементов *MFT* – единственная часть диска, имеющая строго фиксированное положение. Копия этих же 16 записей хранится в середине тома для надежности, поскольку они очень важны. Остальные части *MFT*-файла могут располагаться, как и любой другой файл, в произвольных местах диска – восстановить его положение можно с помощью его самого, «зацепившись» за самую основу – за первый элемент *MFT*.

Упомянутые первые 16 файлов *NTFS* (метафайлы) носят служебный характер; каждый из них отвечает за какой-либо аспект работы системы. Метафайлы находятся в корневом каталоге *NTFS*-тома. Все они начинаются с символа имени «$», хотя получить какую-либо информацию о них стандартными средствами сложно. В табл. 2 приведены основные известные метафайлы и их назначение.

Таблица 2

Метафайлы *NTFS*

Имя метафайла	**Назначение метафайла**
$MFT *$*	Сам *Master File Table*
$MFTmirr	Копия первых 16 записей *MFT*, размещенная посередине тома
$LogFile	Файл поддержки операций журналирования
$Volume	Служебная информация – метка тома, версия файловой системы и т. д.
$AttrDef	Список стандартных атрибутов файлов на томе

$.	Корневой каталог
$Bitmap	Карта свободного места тома
$Boot	Загрузочный сектор (если раздел загрузочный)
$Quota	Файл, в котором записаны права пользователей на использование дискового пространства (этот файл начал работать лишь в *Windows* 2000 с системой *NTFS* 5.0)
$Upcase	Файл – таблица соответствия заглавных и прописных букв в именах файлов. В *NTFS* имена файлов записываются в *Unicode* (что составляет 65 тысяч различных символов) и искать большие и малые эквиваленты в данном случае – нетривиальная задача

Итак, все файлы тома упоминаются в *MFT*. В этой структуре хранится вся информация о файлах, за исключением собственно данных. Имя файла, размер, положение на диске отдельных фрагментов и т. д. – все это хранится в соответствующей записи. Если для информации не хватает одной записи *MFT*, то используется несколько записей, причем не обязательно идущих подряд. Файлы могут иметь не очень большой размер, тогда данные файла хранятся прямо в *MFT*, в оставшемся от основных данных месте в пределах одной записи *MFT*. Файлы, занимающие сотни байт, обычно не имеют своего «физического» воплощения в основной файловой области – все данные такого файла хранятся в одном месте, в *MFT*.

Файл в томе с *NTFS* идентифицируется так называемой файловой ссылкой, которая представляется как 64-разрядное число. Файловая ссылка состоит из номера файла, который соответствует позиции его файловой записи в *MFT*, и номера последовательности. Последний увеличивается всякий раз, когда данная позиция в *MFT* используется повторно, что позволяет файловой системе *NTFS* выполнять внутренние проверки целостности.

Каждый файл в *NTFS* представлен с помощью потоков, то есть у него нет как таковых «просто данных», а есть «потоки». Для правильного понимания потока достаточно указать, что один из потоков и носит привычный нам смысл – данные файла. Но большинство атрибутов файла – это тоже потоки. Таким образом, получается, что базовая сущность у файла только одна – номер в *MFT*, а все остальное, включая и его потоки, – опционально. Данный подход может эффективно использоваться – например, файлу можно «прилепить» еще один поток, записав в него любые данные. В *Windows* 2000 таким образом записана информация об авторе и содержании файла (одна из закладок в свойствах файла, просматриваемых, например, из проводника). Интересно, что эти дополнительные потоки не видны стандартными средствами работы с файлами: наблюдаемый размер файла – это лишь размер основного потока, который содержит традиционные данные. Можно, к примеру, иметь файл нулевой длины, при стирании которого освободится 1 Гбайт свободного места – просто потому, что какая-нибудь хитрая программа или технология «прилепила» к нему дополнительный поток (альтернативные данные) такого большого размера. Но

на самом деле в настоящее время потоки практически не используются, так что опасаться подобных ситуаций не следует, хотя гипотетически они возможны. Просто необходимо иметь в виду, что файл в *NTFS* – это более глубокое понятие, чем можно себе представить, просматривая каталоги диска.

Имя файла в *NTFS*, в отличие от файловой системы *FAT*, может содержать любые символы, включая полный набор национальных алфавитов, так как данные представлены в *Unicode* – 16-битном представлении, которое дает 65535 разных символов. Максимальная длина имени файла в NTFS – 255 символов.

Большой вклад в эффективность файловой системы вносит организация каталога. Каталог в *NTFS* представляет собой специальный файл, хранящий ссылки на другие файлы и каталоги, создавая иерархическое строение данных на диске. Файл каталога поделен на блоки, каждый из которых содержит имя файла, базовые атрибуты и ссылку на элемент *MFT*, который уже предоставляет полную информацию об элементе каталога. Главный каталог диска – корневой – ничем не отличается от обычных каталогов, кроме специальной ссылки на него из начала метафайла *MFT*.

Основные отличия FAT и NTFS

1. Если говорить о накладных расходах на хранение служебной информации, *FAT* отличается от *NTFS* большей компактностью и меньшей сложностью. В большинстве томов *FAT* на хранение таблицы размещения, содержащей информацию обо всех файлах тома, расходуется менее 1 Мбайт. Столь низкие накладные расходы позволяют форматировать в *FAT* жесткие диски малого объема и флоппи-диски. В *NTFS* служебные данные занимают больше места, чем в *FAT*. Так, каждый элемент каталога занимает 2 Кбайт. Однако это имеет и свои преимущества, так как содержимое файлов объемом 1500 байт и менее может полностью храниться в элементе каталога.
2. Система *NTFS* не может использоваться для форматирования флоппи-дисков. Не стоит пользоваться ею для форматирования разделов объемом менее 50 - 100 Мбайт. Относительно высокие накладные расходы приводят к тому, что для малых разделов служебные данные могут занимать до 25% объема носителя. Рекомендуется использовать *FAT* для разделов объемом 256 Мбайт и менее, а *NTFS* - для разделов объемом 400 Мбайт и более.
3. Следующий критерий сравнения – размер файлов. Из-за особенностей своего внутреннего строения разделы *FAT* лучше всего работают для разделов объемом 200 Мбайт и менее. Разделы *NTFS* могут достигать 16 Эбайт, однако в настоящее время из-за аппаратных и других системных причин размер файлов ограничивается 2 Тбайт.
4. Разделы *FAT* могут использоваться практически во всех операционных системах. За редкими исключениями, с разделами *NTFS* можно работать напрямую только из *Windows NT*, хотя и имеются для ряда ОС соответствующие реализации систем управления файлами для чтения файлов из томов *NTFS*.

5. Разделы *FAT* не обеспечивают локальной безопасности. С другой стороны, разделы *NTFS* обеспечивают локальную безопасность как файлов, так и каталогов.

6. *Windows NT* содержит специальную утилиту *CONVERT.EXE*, которая преобразует тома *FAT* в эквивалентные тома *NTFS*, однако для обратного преобразования подобных утилит не существует.

7. В настоящее время *NTFS* рассматривается в качестве предпочтительной файловой системы как для серверных, так и для клиентских версий *Windows*.

Ограничения/ возможности	***NTFS***	***FAT16* и *FAT16***
Размеры диска	2^{64} байт (16 эксабайт или 18 446744 073 709 551 616 байт)	Приблизительно равняется 8 терабайт
Размер тома	Теоретически 2^{64} минус 1 кластер.	С учетом самих таблиц *FAT* и при максимальном размере кластера 32 КБ размер тома может быть до 127,53 ГБ.
Максимальный размер файла	Практически 2^{44} байт минус 64 килобайта (~16384 гигабайт или ~16терабайт)	*FAT*16 поддерживает файлы размером не более 2 ГБ. *FAT*32 поддерживает файлы размером не более 4 ГБ
Поддержка сжатия.	На уровне файловой системы для файлов, каталогов и дисков.	На уровне диска (в FAT16). В *FAT*32 не поддерживается.
Максимальное Количество файлов	4 294 967 295 (2^{32} — 1).	В *FAT*32 не более 268 435 444 (2^{28}–12)

Файловая система *UNIX*

Введение

Впервые система *Unix* была описана в 1974 году в статье Кена Томпсона и Дэнниса Ричи в журнале "Communications of the ACM". С этого времени она получила широкое распространение и завоевала широкую популярность среди производителей ЭВМ, которые все чаще стали оснащать ею свои машины.

Файловая система *Unix* характеризуется иерархической структурой, согласованной обработкой массивов данных, возможностью создания и удаления файлов, динамическим расширением файлов, защитой информации в файлах, трактовкой периферийных устройств (таких как терминалы и ленточные устройства) как файлов.

Файловая система *Unix*

Файловая система в *Unix* – "*деревянная*", состоит из файлов и каталогов. На каждом разделе диска создается собственная независимая файловая система. Отдельные файловые системы "сцепляются" вместе, в единое общее дерево директорий. Такая операция называется "монтированием". Выглядит это примерно так:

mount - F ufs /dev/dsk/m197_c0d0s5 /home1
mount - F ufs /dev/dsk/m197_c0d0s4 /usr
df

Получить доступ к файлам "несмонтированной" файловой системы невозможно. Порочная пракика *MS-DOS*а – сколько разделов, столько и "дисков" (*a: b: c: d: e: ... k: l: m: n*:) в *Unix* не применяется. В *Unix* всегда есть ровно одно общее дерево каталогов, и, по большому счету, пользователям совершенно все равно, на каком именно диске или разделе диска расположены его файлы */usr/spool/moshkow* или */home1/moshkow/bin/mcopy*...

Для повышения производительности дискового ввода/вывода и, соответственно, всей системы в целом, в *Unix* используется кэширование дисковых блоков в памяти. Для этого используется выделенная область оперативной памяти, где кэшируются дисковые блоки файлов, к которым наиболее часто осуществляется доступ. Эта область памяти и связанный с ней процедурный интерфейс носят название буферного кэша, и через него проходит большинство операций файлового ввода/вывода.

Операция записи на диск выполняется не тогда, когда это приказывает выполняемый процесс, а когда операционная система сочтет нужным это сделать. Это резко поднимает эффективность и скорость работы с диском, и повышает опасность ее использования. Выключение питания на "горячей", работающей *Unix*-машине приводит к разрушениям структуры файловой системы.

При каждой начальной загрузке *Unix* проверяет – корректно ли была выключена машина в прошлый раз, и если нет – автоматически запускает утилиту *fsck (File System Check)* – проверку и ремонт файловых систем.

Внутренняя структура файловой системы *Unix*

Каждый жесткий диск состоит из одной или нескольких логических частей, называемых разделами (*partitions*). Расположение и размер раздела определяются при форматировании диска. В *UNIX* разделы выступают в качестве независимых устройств, доступ к которым осуществляется как к различным носителям данных. Например, диск может состоять из четырех разделов, каждый из которых содержит свою файловую систему. Заметим, что в разделе может располагаться только одна файловая система, которая не может занимать несколько разделов. В другой конфигурации диск может состоять только из одного раздела, позволяя создание весьма емких файловых систем.

Раздел диска, в котором создана файловая система, разбит на три части:

1. СУПЕРБЛОК. Занимает 1 *Kb*. Содержит служебную информацию:
Тип файловой системы. Размер. Начало списка свободных блоков. Что-то еще.

2. ОБЛАСТЬ *INOD*-ов. Занимает примерно 8% общего размера раздела. *INODE (Index-node)* – описатель файла. Он содержит всю информацию о файле, за исключением имени файла, и собственно данных файла. В *INODE* хранится:

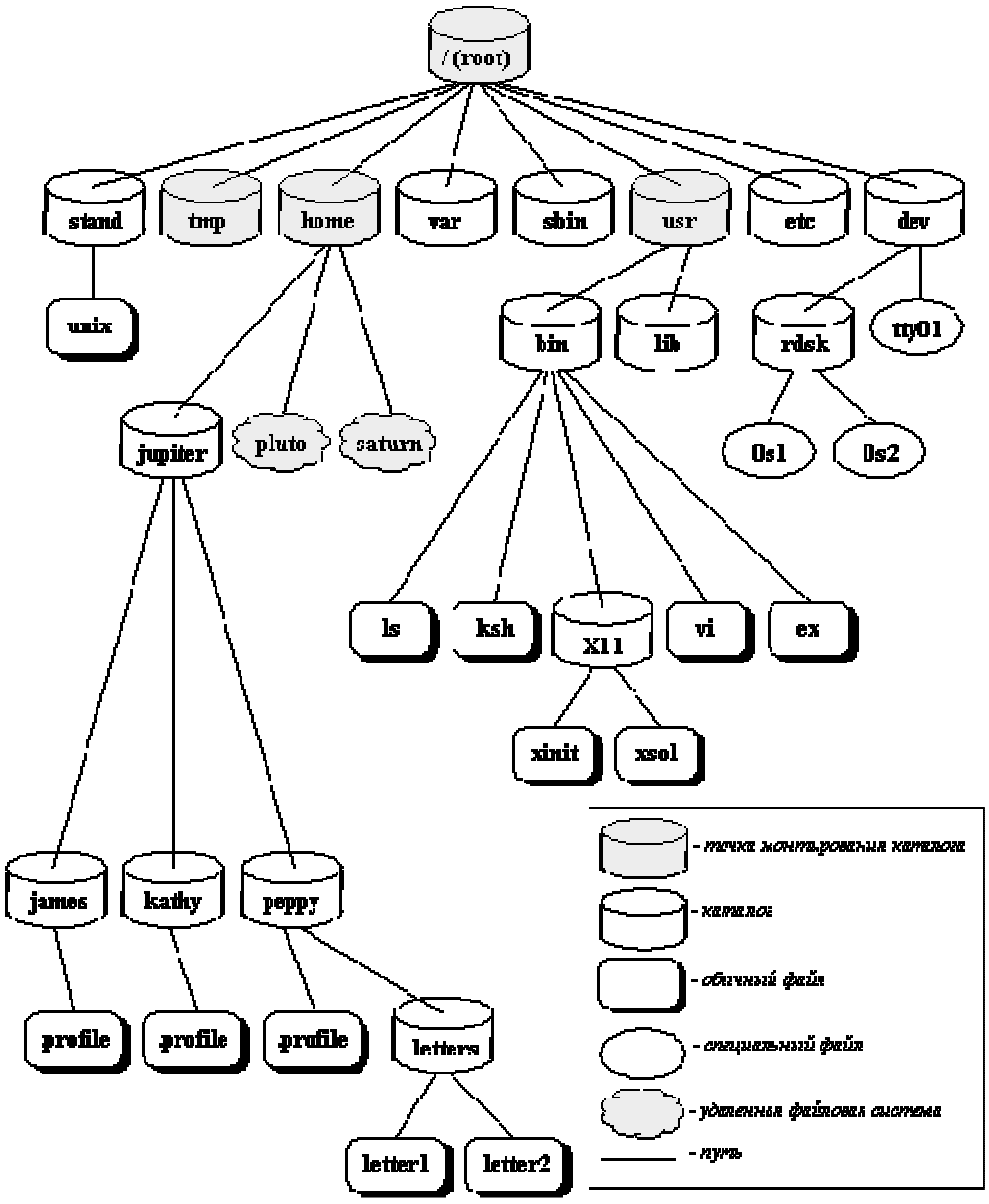
тип файла (файл, каталог, именованный канал, специальный файл) кто владелец права (атрибуты) файла время модификации/создания файла адреса блоков, из которых состоит файл, что-то еще...

3. ОБЛАСТЬ ДАННЫХ. В этой области расположены блоки с данными файлов. Незанятые блоки провязаны в СПИСОК СВОБОДНЫХ БЛОКОВ.

Файлы бывают двух основных типов. ФАЙЛ, КАТАЛОГ.
ФАЙЛ – он и есть файл.
КАТАЛОГ – файл фиксированного формата: состоит из строчек с именами файлов, входящих в каталог:
имя_файла1 Номер_Инода1
имя_файла2 Номер_Инода2

Чтобы получить доступ к файлу по имени, операционная система находит это имя в каталоге, содержащем файл, берет Номер_Инода файла, по номеру находит *inod* в области *inod*'ов, из *inod*'а берет адреса блоков, в которых расположены данные файла, по адресам блоков считывает блоки из области данных.

В мире ОС *UNIX* по историческим причинам термин "файловая система" является перегруженным, обозначая одновременно иерархию каталогов и файлов и часть ядра, которая управляет каталогами и файлами. Видимо, было бы правильнее называть иерархию каталогов и файлов архивом файлов, а термин "файловая система" использовать только во втором смысле. Каждый каталог и файл файловой системы имеет уникальное полное имя (в ОС UNIX это имя принято называть *full pathname* – имя, задающее полный путь, поскольку оно действительно задает полный путь от корня файловой системы через цепочку каталогов к соответствующему каталогу или файлу; мы будем использовать термин "полное имя", поскольку для *pathname* отсутствует благозвучный русский аналог). Каталог, являющийся корнем файловой системы (корневой каталог), в любой файловой системе имеет предопределенное имя "/" (слэш). Полное имя файла, например, */bin/sh* означает, что в корневом каталоге должно содержаться имя каталог*a bin*, а в каталоге *bin* должно содержаться имя файла *sh*. Коротким или относительным именем файла (*relative pathname*) называется имя (возможно, составное), задающее путь к файлу от текущего рабочего каталога (существует команда и соответствующий системный вызов, позволяющие установить текущий рабочий каталог).



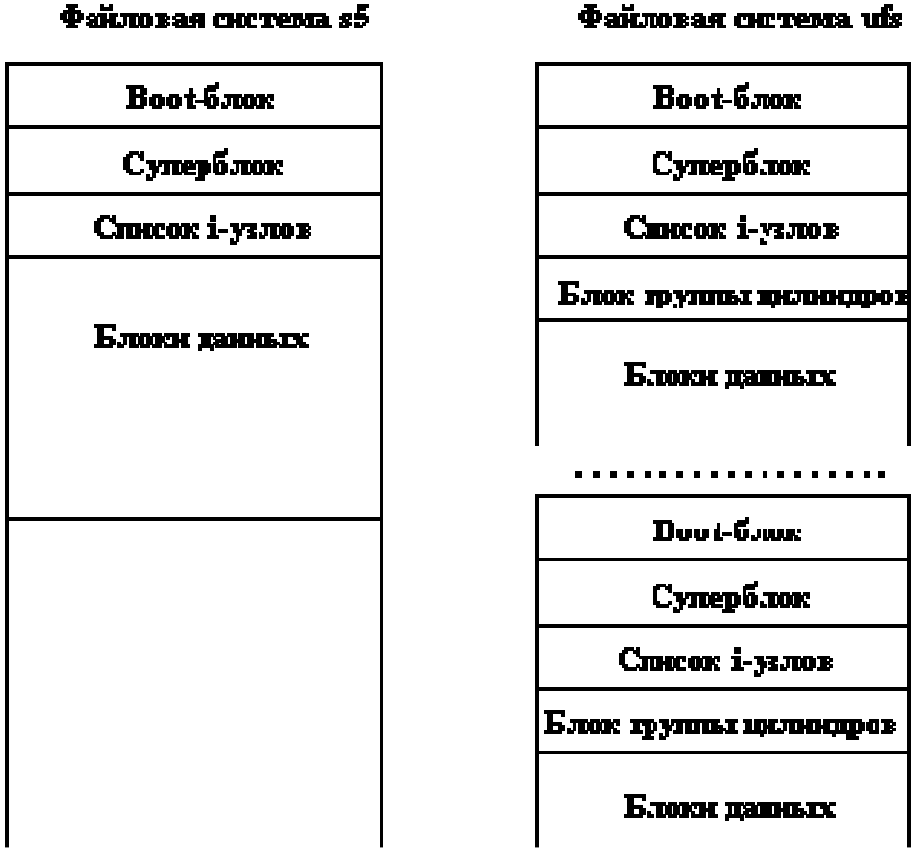
Структура каталогов файловой системы *UNIX* поддерживает многочисленные утилиты, позволяющие работать с файловой системой и доступные как команды командного интерпретатора. Вот некоторые из них (наиболее употребительные):

cp имя1 имя2	- копирование файла имя1 в файл имя2
rm имя1	- уничтожение файла имя1
mv имя1 имя2	- переименование файла имя1 в файл имя2
mkdir имя	- создание нового каталога имя
rmdir имя	- уничтожение каталога имя
ls имя	- выдача содержимого каталога имя
cat имя	- выдача на экран содержимого файла имя
chown имя режим	- изменение режима доступа к файлу

Структура файловой системы

Файловая система обычно размещается на дисках или других устройствах внешней памяти, имеющих блочную структуру. Кроме блоков, сохраняющих каталоги и файлы, во внешней памяти поддерживается еще несколько служебных областей.

В мире *UNIX* существует несколько разных видов файловых систем со своей структурой внешней памяти. Наиболее известны традиционная файловая система *UNIX System V (s5fs)* и файловая система семейства *UNIX BSD (ufs)*. Файловая система *s5fs* состоит из четырех секций. В файловой системе *ufs* на логическом диске (разделе реального диска) находится последовательность секций файловой системы.



Исконной файловой системой *UNIX System V* является *s5fs*. Файловая система, разработанная в Беркли, *ufs*, появилась позже, в версии 4.2*BSD UNIX*. По сравнению с *s5fs* она обладает лучшей производительностью, функциональностью и надежностью.

Кратко опишем суть и назначение каждой области диска.

***Boot*-блок** содержит программу раскрутки, которая служит для первоначального запуска ОС *UNIX*. В файловых системах *s5fs* реально

используется *boot*-блок только корневой файловой системы. В дополнительных файловых системах эта область присутствует, но не используется.

Суперблок – это наиболее ответственная область файловой системы, содержащая информацию, которая необходима для работы с файловой системой в целом. Суперблок содержит список свободных блоков и свободные *i*-узлы (*information nodes* – информационные узлы). В файловых системах *ufs* для повышения устойчивости поддерживается несколько копий суперблока (по одной копии на группу цилиндров). Каждая копия суперблока имеет размер 8196 байт, и только одна копия суперблока используется при монтировании файловой системы. Однако, если при монтировании устанавливается, что первичная копия суперблока повреждена или не удовлетворяет критериям целостности информации, используется резервная копия.

Блок группы цилиндров содержит число *i*-узлов, специфицированных в списке *i*-узлов для данной группы цилиндров, и число блоков данных, которые связаны с этими *i*-узлами. Размер блока группы цилиндров зависит от размера файловой системы. Для повышения эффективности файловая система *ufs* старается размещать *i*-узлы и блоки данных в одной и той же группе цилиндров.

Список *i*-узлов (*ilist*) содержит список *i*-узлов, соответствующих файлам данной файловой системы. Максимальное число файлов, которые могут быть созданы в файловой системе, определяется числом доступных *i*-узлов. В *i*-узле хранится информация, описывающая файл: режимы доступа к файлу, время создания и последней модификации, идентификатор пользователя и идентификатор группы создателя файла, описание блочной структуры файла и т.д.

Блоки данных – в этой части файловой системы хранятся реальные данные файлов. В случае файловой системы *ufs* все блоки данных одного файла пытаются разместить в одной группе цилиндров. Размер блока данных определяется при форматировании файловой системы командой *mkfs* и может быть установлен в 512, 1024, 2048, 4096 или 8192 байтов.

Рассмотрим подробнее каждый из перечисленных компонентов.

Суперблок

Суперблок содержит информацию, необходимую для монтирования и управления работой файловой системы в целом (например, для размещения новых файлов). В каждой файловой системе существует только один суперблок, который располагается в начале раздела. Суперблок считывается в память при монтировании файловой системы и находится там до ее отключения (размонтирования). Суперблок содержит следующую информацию:

- Тип файловой системы (s_type).
- Размер файловой системы в логических блоках, включая сам суперблок, *ilist* и блоки хранения данных (*s_fsize*).
- Размер массива индексных дескрипторов (*s_isize*).

- Число свободных блоков, доступных для размещения (*s_tfree*).
- Число свободных *inode*, доступных для размещения (*s_tinode*).
- Флаги (флаг *модификации s_fmod*, флаг режима монтирования *s_fronly*).
- Размер логического блока.
- Список номеров свободных *inode*.
- Список адресов свободных блоков.

Выделение свободных блоков для размещения файла производится с конца списка суперблока. Когда в списке остается единственный элемент, ядро интерпретирует его как указатель на блок, содержащий продолжение списка. В этом случае содержимое этого блока считывается в суперблок и блок становится свободным. Такой подход позволяет использовать дисковое пространство под списки, пропорциональное свободному месту в файловой системе. Другими словами, когда свободного места практически не остается список адресов свободных блоков целиком помещается в суперблоке.

Индексные дескрипторы

Индексный дескриптор, или *inode*, содержит информацию о файле, необходимую для обработки данных, т. е. метаданные файла. Каждый файл ассоциирован с одним *inode*, хотя может иметь несколько имен в файловой системе, каждое из которых указывает на один и тот же *inode*.

Индексный дескриптор не содержит:

- имени файла, которое хранится в файлах специального типа – каталогах;
- содержимого файла, которое размещено в блоках хранения данных.

При открытии файла ядро помещает копию дискового *inode* в таблицу *in-core inode*, которая содержит несколько дополнительных полей. Основные поля дискового *inode* следующие:

di_mode	Тип файла, дополнительные атрибуты выполнения и права доступа.
di_nlinks	Число ссылок на файл, т. е. количество имен, которые имеет файл в файловой системе.
di_uid, di_gid	Идентификаторы владельца-пользователя и владельца-группы.
di_size	Размер файла в байтах. Для специальных файлов это поле содержит старший и младший номера устройства.
di_atime	Время последнего доступа к файлу.

di_mtime	Время последней модификации.
di_ctime	Время последней модификации inode (кроме модификации полей di_atime, dijntime).
*di_addr[*13*]*	Массив адресов дисковых блоков хранения данных.

Заметим, что в индексном дескрипторе отсутствует информация о времени создания файла. Вместо этого *inode* хранит три значения времени последнего доступа (*di_atime*), время последней модификации содержимого файла (*di_mtime*) и время последней модификации метаданных файла (*di_ctime*). В последнем случае не учитываются модификации полей *di_atime* и *di_mtime*. Таким образом, *di_ctime* изменяется, когда изменяется размер файла, владелец, группа, или число связей.

Индексный дескриптор содержит информацию о расположении данных файла. Поскольку дисковые блоки хранения данных файла в общем случае располагаются не последовательно, *inode* должен хранить физические адреса всех блоков, принадлежащих данному файлу. В индексном дескрипторе эта информация хранится в виде массива, каждый элемент которого содержит физический адрес дискового блока, а индексом массива является номер логического блока файла. Массив имеет фиксированный размер и состоит из 13 элементов. При этом первые 10 элементов адресуют непосредственно блоки хранения данных файла. Одиннадцатый элемент адресует блок, в свою очередь содержащий адреса блоков хранения данных. Двенадцатый элемент указывает на дисковый блок, также хранящий адреса блоков, каждый из который адресует блок хранения данных файла. И, наконец, тринадцатый элемент используется для тройной косвенной адресации, когда для нахождения адреса блока хранения данных файла используются три дополнительных блока.

Такой подход позволяет при относительно небольшом фиксированном размере индексного дескриптора поддерживать работу с файлами, размер которых может изменяться от нескольких байтов до десятка мегабайт. Для относительно небольших файлов (до 10 Кбайт при размере блока 1024 байтов) используется прямая индексация, обеспечивающая максимальную производительность. Для файлов, размер которых не превышает 266 Кбайт (10 Кбайт + 256x1024), достаточно простой косвенной адресации. Наконец, при использовании тройной косвенной адресации можно обеспечить доступ к 16777216 блокам (256x256x256).

Имена файлов

Как мы уже видели, ни метаданные, ни тем более блоки хранения данных, не содержат имени файла.

Имя файла хранится в файлах специального типа – каталогах. Такой подход позволяет любому файлу, т. е. фактическим данным, иметь теоретически неограниченное число имен (названий), в файловой системе. При этом несколько имен файлов будут соответствовать одним и тем же метаданным и данным и являться жесткими связями.

Каталог файловой системы *s5fs* представляет собой таблицу, каждый элемент которой имеет фиксированный размер в 16 байтов: 2 байта хранят номер индексного дескриптора файла, а 14 байтов – его имя. Это накладывает ограничение на число *inode*, которое не может превышать 65535. Также ограничена и длина имени файла: его максимальный размер – 14 символов.

Первые два элемента каталога адресуют сам каталог (текущий каталог) под именем "." и родительский каталог под именем "..".

При удалении имени файла из каталога номер *inode* соответствующего элемента устанавливается равным 0. Ядро обычно не удаляет такие свободные элементы, поэтому размер каталога не уменьшается даже при удалении файлов. Это является потенциально проблемой для каталогов, в которые временно было помещено большое количество файлов. После удаления большинства из них размер каталога останется достаточно большим, поскольку записи удаленных файлов будут по-прежнему существовать.

Управление задачами в операционных системах

Операционная система выполняет следующие основные функции, связанные с управлением процессами и задачами:

- создание и удаление задач;
- планирование процессов и диспетчеризация задач;
- синхронизация задач, обеспечение их средствами коммуникации.

Система управления процессами обеспечивает прохождение процесса через компьютер. В зависимости от состояния процесса ему должен быть предоставлен тот или иной ресурс. Например, новый процесс необходимо разместить в основной памяти, следовательно, ему необходимо выделить часть адресного пространства. Процессу в состоянии готовый должно быть предоставлено процессорное время. Выполняемый процесс может потребовать оборудование ввода-вывода и доступ к файлу.

Очереди. Распределение процессов между имеющимися ресурсами носит название *планирование процессов.* Одним из методов планирования процессов, ориентированных на эффективную загрузку ресурсов, является метод очередей ресурсов. Новые процессы находятся во входной очереди, часто называемой *очередью работ – заданий.*

Входная очередь располагается во внешней памяти, во входной очереди процессы ожидают освобождения ресурса – адресного пространства основной памяти.

Готовые к выполнению процессы располагаются в основной памяти и связаны *очередью готовых процессов.* Процессы в этой очереди ожидают освобождения ресурса *процессорное время.*

Процесс в состоянии ожидания завершения операции ввода-вывода находится в одной из *очередей к оборудованию* ввода-вывода.

При прохождении через компьютер процесс мигрирует между различными очередями под управлением программы, которая называется *планировщик* (*scheduler*).

Операционная система, обеспечивающая режим мультипрограммирования, обычно включает два планировщика – *долгосрочный* и *краткосрочный.* Например, в OS/360 долговременный планировщик назывался *планировщиком заданий,* а краткосрочный – *супервизором задач.*

На уровень долгосрочного планирования выносятся редкие системные действия, требующие больших затрат системных ресурсов, на уровень краткосрочного планирования – частые и более короткие процессы. На каждом уровне существует свой объект и собственные средства управления им.

Основное различие между долгосрочным и краткосрочным планировщиками заключается в частоте запуска, например, краткосрочный планировщик может запускаться каждые 100 мс, долгосрочный – 1 раз за несколько минут.

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых процессов в случае освобождения ресурсов памяти.

Долгосрочный планировщик выбирает процесс из входной очереди с целью создания неоднородной мультипрограммной смеси. Это означает, что в очереди готовых процессов должны находиться в разной пропорции как процессы, ориентированные на ввод-вывод, так и процессы, ориентированные на преимущественную работу с *CPU*.

Краткосрочный планировщик решает, какой из процессов, находящихся в очереди готовых процессов, должен быть передан на выполнение в *CPU*. В некоторых операционных системах долгосрочный планировщик может отсутствовать. Например, в *системах разделения времени* (*time-sharing system*) каждый новый процесс сразу же помещается в основную память.

На уровне *краткосрочного* планирования объектом управления являются процессы, которые выступают как потребители центрального процессора для внутренних процессов или внешнего процессора для внешних процессов. Причинами порождения процесса могут быть процессы на том же уровне или сигналы, посылаемые от долгосрочного планировщика.

Выделение процессора процессу производится многократно, с целью достижения эффекта мультипрограммирования, и такой процесс называется *диспетчеризацией*.

Планирование процессов и диспетчеризация задач

1. Стратегии диспетчеризации

Стратегия диспетчеризации определяет, какие процессы планируются на выполнение, чтобы достичь поставленной цели. Существует много стратегий, можно назвать некоторые из них:

- по возможности заканчивать вычислительные процессы в том же самом порядке, в котором они были начаты;
- отдавать предпочтение более коротким процессам;
- предоставлять всем процессам пользователей одинаковые услуги, в том числе и одинаковое время ожидания.

Когда говорят о стратегии обслуживания, всегда имеют в виду понятие процесса, поскольку процесс может состоять из нескольких задач.

2. Дисциплины диспетчеризации

Известно большое количество правил (дисциплин) диспетчеризации, в соответствии с которыми формируется список (очередь) готовых к выполнению задач. Различают два больших класса таких дисциплин – *бесприоритетные* и

приоритетные. При реализации приоритетных дисциплин отдельным задачам предоставляется преимущественное право попасть в состояние исполнения.

Запомним о приоритетах следующее:

- приоритет, присвоенный задаче, может являться величиной постоянной;
- приоритет задачи может изменяться в процессе ее решения.

Диспетчеризация с динамическими приоритетами требует дополнительных расходов на вычисление значений приоритетов исполняющихся задач, поэтому во многих ОС реального времени используются методы диспетчеризации на основе статических (постоянных) приоритетов. Хотя надо заметить, что динамические приоритеты позволяют реализовать гарантии обслуживания задач.

При бесприоритетном обслуживании выбор задачи производится в некотором порядке без учета их важности и времени обслуживания. При реализации приоритетных дисциплин обслуживания отдельным задачам предоставляется преимущественное право на исполнение.

Бесприоритетные дисциплины обслуживания делятся на: (рис. 1.)

- ***линейные:***
  - в порядке очереди;
  - случайный выбор процесса;

- ***циклические:***
  - циклический алгоритм;
  - многоуровневый циклический алгоритм.

Приоритетные дисциплины обслуживания делятся на: (рис. 1.)

- ***с фиксированным приоритетом:***
  - с относительным приоритетом;
  - с абсолютным приоритетом;
  - адаптивное обслуживание;
  - приоритет зависит от времени ожидания;

- ***с динамическим приоритетом:***
  - приоритет зависит от времени ожидания;
  - приоритет зависит от времени обслуживания.

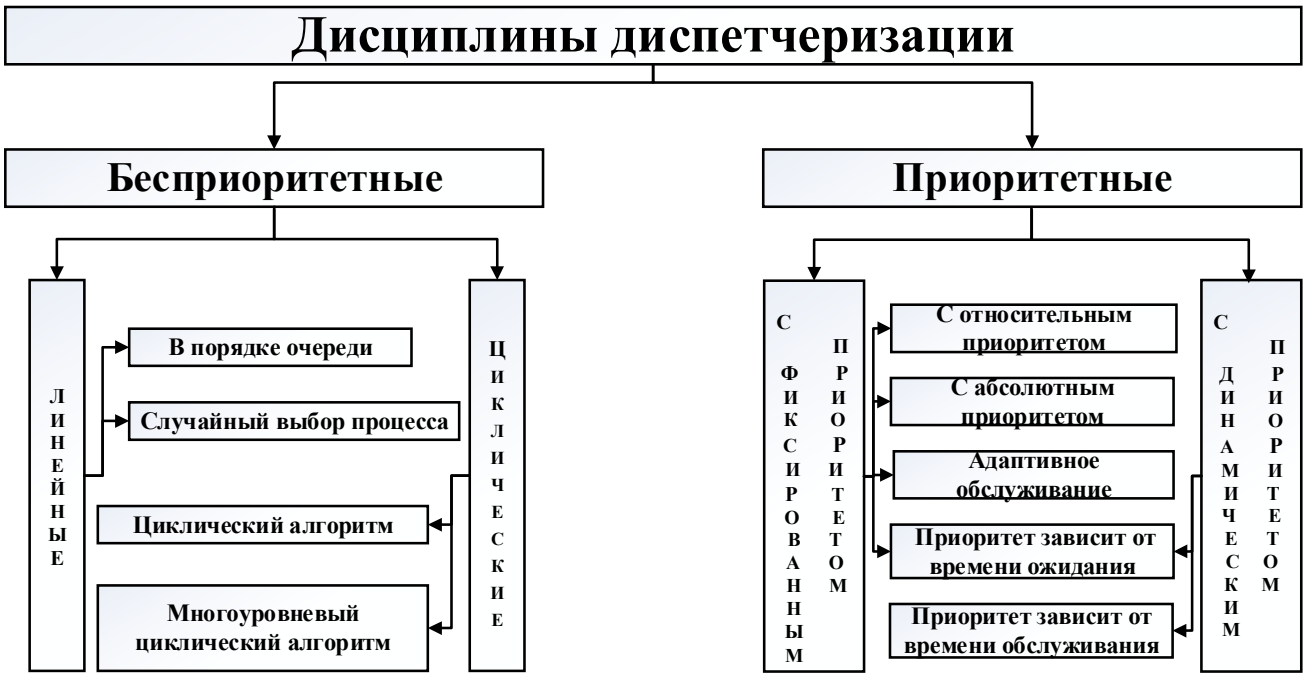


Рис.1. Дисциплины диспетчеризации

Рассмотрим кратко некоторые наиболее часто используемые дисциплины диспетчеризации.

Самой простой в реализации является *дисциплина FCFS* (*first come – first served*), согласно которой задачи обслуживаются в порядке очереди, т.е. в порядке их появления (рис. 2). Те задачи, которые были заблокированы в процессе работы, после перехода в состояние готовности ставятся в эту очередь перед теми задачами, которые еще не выполнялись, т.е. образуется две очереди – одна из новых задач, а вторая очередь – из ранее выполнявшихся, но попавших в состояние ожидания. Такой подход позволяет реализовать стратегию обслуживания «по возможности заканчивать вычислительные процессы в порядке их появления».

Эта дисциплина обслуживания не требует внешнего вмешательства в ход вычислений, при ней не происходит перераспределение процессорного времени. Существующие дисциплины диспетчеризации процессов могут быть разбиты на два класса – вытесняющие (*preemptive*) и не вытесняющие (*non-preemptive*). В первых пакетных ОС часто реализовывали параллельное выполнение заданий без принудительного перераспределения процессора между задачами. В большинстве современных ОС для мощных вычислительных систем, а также и в ОС для ПК, ориентированных на высокопроизводительное выполнение приложений (*Windows NT*, *OS/2*, *Linux*), реализована вытесняющая многозадачность. Можно сказать, что рассмотренная дисциплина относится к не вытесняющим.

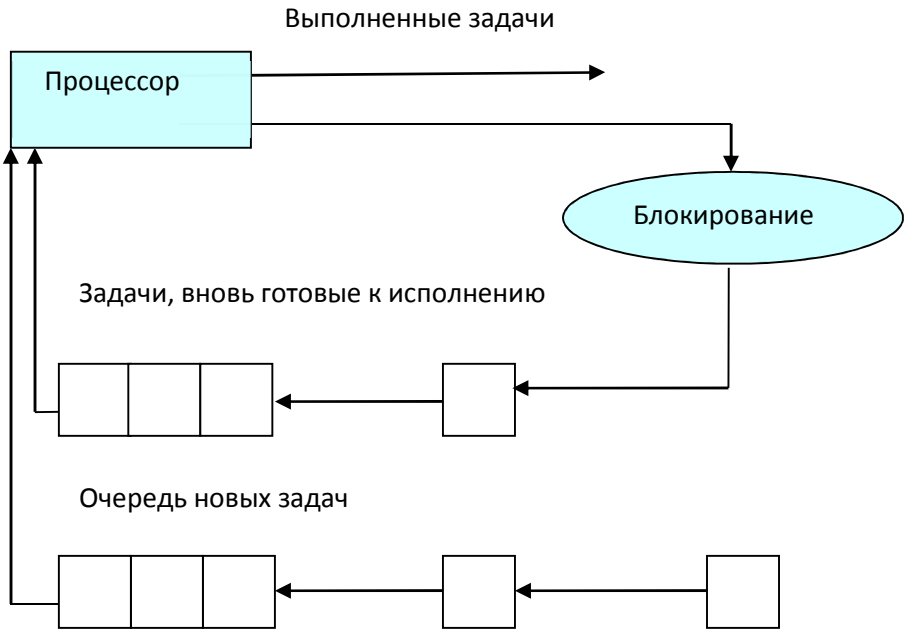


Рис.2. Дисциплина *FCFS*

К достоинствам этой дисциплины можно отнести простоту реализации и малые расходы системных ресурсов на формирование очереди задач.

Но она приводит к тому, что при увеличении загрузки вычислительной системы растет и среднее время ожидания обслуживания, причем короткие задания вынуждены ожидать столько же, сколько и трудоемкие задания. Избежать этого недостатка позволяют дисциплины *SJN* и *SRT*.

Дисциплина обслуживания *SJN* (*shortest job next*, что означает: следующим будет выполняться кратчайшее задание) требует, чтобы для каждого задания была известна оценка в потребностях машинного времени. Необходимость сообщать ОС характеристики задач, в которых описывались бы потребности в ресурсах вычислительной системы, привела к тому, что были разработаны соответствующие языковые средства. В частности, язык *JCL* (*job control language*, язык управления заданиями) был одним из наиболее известных. Пользователи вынуждены были указывать предполагаемое время выполнения, и для того, чтобы они не злоупотребляли возможностью указать заведомо меньшее время выполнения (с целью получить результаты раньше других), ввели подсчет реальных потребностей. Диспетчер задач сравнивал заказанное время и время выполнения, и в случае превышения указанной оценки в данном ресурсе ставил данное задание не в начало, а в конец очереди. Еще в некоторых ОС в таких случаях использовалась система штрафов, при которой в случае превышения заказанного машинного времени оплата вычислительных ресурсов осуществлялась уже по другим расценкам.

Дисциплина обслуживания SJN предполагает, что имеется только одна очередь заданий, готовых к выполнению. И задания, которые в процессе своего исполнения были временно заблокированы (например, ожидали завершения

операций ввода/вывода), вновь попадают в конец очереди готовых к выполнению наравне с вновь поступающими. Это приводит к тому, что задания, которым требуется очень немного времени для своего завершения, вынуждены ожидать процессор наравне с длительными работами, что не всегда хорошо.

Для устранения этого недостатка и была предложена дисциплина *SRT* (*shortest remaining time*, следующее задание требует меньше всего времени для своего завершения).

Приоритетное планирование. Описанные ранее стратегии могут рассматриваться как частные случаи стратегии приоритетного планирования. Эта стратегия предполагает, что каждому процессу приписывается приоритет, определяющий очередность предоставления ему *CPU*. Например, стратегия *FCFS* предполагает, что все процессы имеют одинаковые приоритеты, а стратегия *SJN* предполагает, что приоритет есть величина, обратная времени последующего обслуживания.

Обычно приоритет – это целое положительное число, находящееся в некотором диапазоне, например, от 0 до 7 или от 0 до 1024. Будем считать, что чем меньше значение числа, тем выше приоритет процесса. Приоритеты назначаются, исходя из совокупности внутренних и внешних по отношению к операционной системе факторов.

Внутренние факторы:

• требования к памяти;

• количество открытых файлов;

• отношение среднего времени ввода-вывода к среднему времени использования ресурсов *CPU* и т. д.

Внешние факторы:

• важность процесса;

• тип и величина файлов, используемых для оплаты;

• отделение, выполняющее работы, и т. д.

Главный недостаток приоритетного планирования заключается в возможности блокирования на неопределенно долгое время низкоприоритетных процессов.

Известен случай, когда в 1973 г. в Массачусетском технологическом институте при остановке компьютера *IBM* 7094 в очереди готовых процессов были обнаружены процессы, активизированные в 1967 г., но так и не выполненные.

Все эти три дисциплины обслуживания могут использоваться для пакетных режимов обработки, когда пользователь не вынужден ожидать реакции системы, а просто сдает свое задание и через несколько часов получает свои результаты вычислений. Для интерактивных же вычислений желательно прежде всего обеспечить приемлемое время реакции системы и равенство в обслуживании, если система является мультитерминальной. Если же это однопользовательская система, но с возможностью мультипрограммной обработки, то желательно, чтобы те программы, с которыми мы сейчас непосредственно работаем, имели лучшее

время реакции, нежели наши фоновые задания. При этом мы можем пожелать, чтобы некоторые приложения, выполняясь без нашего непосредственного участия (например, программа получения электронной почты, использующая модем и коммутируемые линии для передачи данных), тем не менее гарантированно получали необходимую им долю процессорного времени. Для решения подобных проблем используется дисциплина обслуживания, называемая *RR* (*round robin*, круговая, карусельная), и приоритетные методы обслуживания.

Дисциплина RR (рис.3) предполагает, что каждая задача получает процессорное время порциями (квантами времени $q = 10\text{-}100\ ms$). После окончания кванта времени q задача снимается с процессора, и он передается следующей задаче. Снятая задача ставится в конец очереди задач, готовых к исполнению. Для оптимальной работы системы необходимо правильно выбрать закон, по которому кванты времени выделяются задачам.

Величина кванта времени q выбирается как компромисс между приемлемым временем реакции системы на запросы пользователей (с тем, чтобы их простейшие запросы не вызывали длительного ожидания) и накладными расходами на частую смену контекста задач. Очевидно, что при прерываниях ОС вынуждена сохранить достаточно большой объем информации о текущем (прерываемом) процессе, поставить дескриптор снятой задачи в очередь, загрузить контекст задачи, которая теперь будет выполняться (ее дескриптор был первым в очереди готовых к исполнению). Если величина q велика, то при увеличении очереди готовых к выполнению задач реакция системы станет плохой. Если же величина мала, то относительная доля накладных расходов на переключения между исполняющимися задачами станет большой и это ухудшит производительность системы. В некоторых ОС есть возможность указывать в явном виде величину q либо диапазон ее возможных значений, поскольку система будет стараться выбирать оптимальное значение сама.

Дисциплина диспетчеризации *RR* – это одна из самых распространенных дисциплин. Однако бывают ситуации, когда ОС не поддерживает в явном виде дисциплину карусельной диспетчеризации. Например, в некоторых ОС реального времени используется диспетчер задач, работающий по принципам абсолютных приоритетов (процессор предоставляется задаче с максимальным приоритетом, а при равенстве приоритетов он действует по принципу очередности). Другими словами, снять задачу с выполнения может только появление задачи с более высоким приоритетом. Поэтому если нужно организовать обслуживание задач таким образом, чтобы все они получали процессорное время равномерно и равноправно, то системный оператор может сам организовать эту дисциплину. Для этого достаточно всем пользовательским задачам присвоить одинаковые приоритеты и создать одну высокоприоритетную задачу, которая не должна ничего делать, но которая, тем не менее, будет по таймеру (через указанные интервалы времени) планироваться на выполнение. Эта задача снимет с выполнения текущее приложение, оно будет поставлено в конец очереди, и поскольку этой высокоприоритетной задаче на самом деле

ничего делать не надо, то она тут же освободит процессор и из очереди готовности будет взята следующая задача.

В своей простейшей реализации дисциплина карусельной диспетчеризации предполагает, что все задачи имеют одинаковый приоритет. Если же необходимо ввести механизм приоритетного обслуживания, то это, как правило, делается за счет организации нескольких очередей. Процессорное время будет предоставляться в первую очередь тем задачам, которые стоят в самой привилегированной очереди. Если она пустая, то диспетчер задач начнет просматривать остальные очереди. Именно по такому алгоритму действует диспетчер задач в операционных системах *OS*/2 и *Windows NT*.

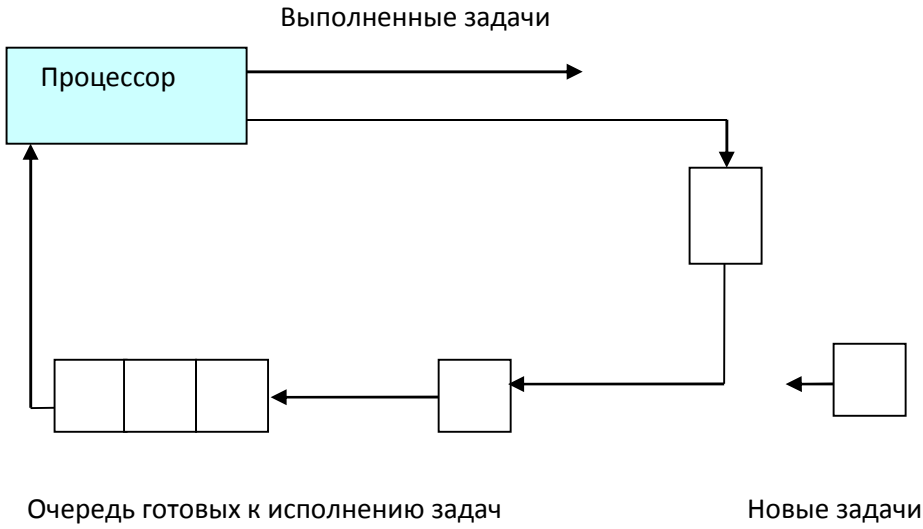


Рис.3. Карусельная дисциплина диспетчеризации *RR*

Диспетчеризация без перераспределения процессорного времени, т.е. *не вытесняющая многозадачность* – это такой способ диспетчеризации процессов, при котором активный процесс выполняется до тех пор, пока он сам не отдаст управление диспетчеру задач. Дисциплины *FCFS, SJN, SRT* относятся к не вытесняющим.

Диспетчеризация с перераспределением процессорного времени между задачами, т.е. *вытесняющая многозадачность* – это такой способ, при котором решение о переключении с одного процесса на другой принимается диспетчером задач, а не самой активной задачей. Дисциплина *RR* относится к вытесняющим.

Для обеспечения еще более быстрой реакции системы на короткие работы в системах оперативной обработки используются *алгоритмы многоуровневого циклического планирования*. Одним из таких алгоритмов является алгоритм *FB (foreground-background).* Принцип его работы можно проиллюстрировать следующей схемой:

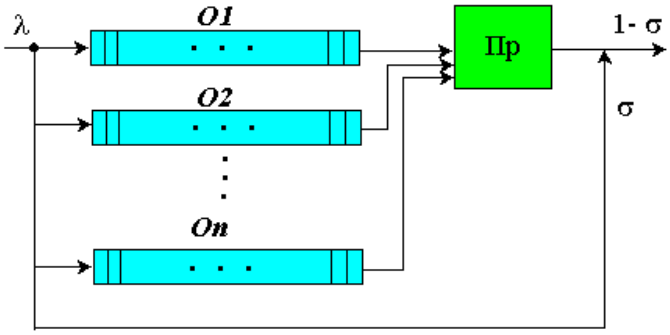


Рис.4. Многоуровневый циклический алгоритм *FB*

Заявки на выполнение работ поступают в очередь *O*1. Работы, стоящие в очереди *O1*, получают квант процессорного времени *q*. Если за это время работа была выполнена, то она покидает систему. В противном случае заявка на работу переносится в очередь *O*2, откуда она может быть занесена в очереди *O*3, *O*4, ..., On. Очереди обслуживаются в следующем порядке. Если имеется хотя бы одна заявка в очереди *O1*, то эта заявка непременно обслуживается. Заявки из очереди *O*2 обслуживаются при условии, что нет заявок в очереди *O1*. Аналогично заявки из очереди *Om* обслуживаются только в том случае, если все очереди *O*1, ..., *Om* пусты. Заявка, достигшая последней очереди *On*, остается в ней до полного завершения работы.

Применяются модификации алгоритма *FB*, различающиеся по величине квантов времени, предоставляемых заявкам из разных очередей. Возможно планирование на основе постоянной величины кванта или с использованием квантов переменной длительности, которая возрастает по мере увеличения номера очереди. Одна из таких модификаций – алгоритм планирования *FB* с учетом приоритетов работ. Работы, поступающие в систему, разделяются в зависимости от приоритетов 1, ..., *n* на *n* потоков. Приоритеты задач относительны, т.е. поступление в систему заявки более высокого приоритета не прерывает процесс обработки менее приоритетных заявок, но при освобождении ресурса более приоритетные заявки будут назначены в первую очередь. Работы с высшим приоритетом поступают в очередь *O1*, а работы с низким приоритетом – в очередь *On*. Работам, выбираемым на обслуживание из разных очередей, выделяются кванты времени различной длительности, причем заявкам из очереди *Om* выделяется больший по продолжительности квант времени, чем заявкам из очереди *Om*-1, *m*=2, …, *n*.

Известны следующие критерии, позволяющие сравнивать алгоритмы краткосрочных планировщиков:

• *утилизация CPU* (использование процессора). Утилизация *CPU* теоретически может находиться в пределах от 0 до 100 %. В реальных системах утилизация *CPU* колеблется в пределах 40 % для легко загруженного *CPU*, 90 % для тяжело загруженного *CPU*;

• *пропускная способность CPU*. Пропускная способность *CPU* может измеряться количеством процессов, которые выполняются в единицу времени;

• *время оборота*. Для некоторых процессов важным критерием является полное время выполнения, т. е. интервал от момента появления процесса во входной очереди до момента его завершения. Это время названо временем оборота и включает время ожидания во входной очереди, время ожидания в очереди готовых процессов, время ожидания в очередях к оборудованию, время выполнения в процессоре и время ввода-вывода;

• *время ожидания* – под этим понимается суммарное время нахождения процесса в очереди готовых процессов;

• *время отклика* – для интерактивных программ важным показателем является время отклика или время, прошедшее от момента попадания процесса во входную очередь до момента первого обращения к терминалу.

Очевидно, что простейшая стратегия краткосрочного планировщика должна быть направлена на максимизацию средних значений загруженности и пропускной способности, времени ожидания и времени отклика.

3. Диспетчеризация задач с использованием динамических приоритетов

При выполнении программ, реализующих какие-либо задачи контроля и управления (что характерно, прежде всего, для систем реального времени), может случиться такая ситуация, когда одна или несколько задач не могут быть реализованы (решены) в течение длительного промежутка времени из-за возросшей нагрузки в вычислительной системе. Потери, связанные с невыполнением таких задач, могут оказаться больше, чем потери от невыполнения программ с более высоким приоритетом. При этом оказывается целесообразным временно изменить приоритет «аварийных» задач (для которых истекает отпущенное для них время обработки). После выполнения этих задач их приоритет восстанавливается. Поэтому почти в любой ОС реального времени имеются средства для изменения приоритета программ. Есть такие средства и во многих ОС, которые не относятся к классу ОСРВ. Введение механизмов динамического изменения приоритетов позволяет реализовать более быструю реакцию системы на короткие запросы пользователей, что очень важно при интерактивной работе, но при этом гарантировать выполнение любых запросов.

Например, в *Windows NT* каждый поток (тред) имеет базовый уровень приоритета, который лежит в диапазоне от двух уровней ниже базового приоритета процесса, его породившего, до двух уровней выше этого приоритета, как показано на рис. 4. Базовый приоритет процесса определяет, сколь сильно могут различаться приоритеты потоков процесса и как они соотносятся с приоритетами потоков других процессов. Поток наследует этот базовый приоритет и может изменять его так, чтобы он стал немного больше или немного меньше. В результате получается приоритет планирования, с которым поток и

начинает исполняться. В процессе исполнения потока его приоритет может отклоняться от базового.

На рис. 4 показан динамический приоритет потока, нижней границей которого является базовый приоритет потока, а верхняя – зависит от вида работ, исполняемых потоком. Например, если поток обрабатывает пользовательский ввод, то диспетчер задач *Windows NT* поднимает его динамический приоритет; если же он выполняет вычисления, то диспетчер постепенно снижает его приоритет до базового. Снижая приоритет одного процесса и поднимая приоритет другого, подсистемы могут управлять относительной приоритетностью потоков внутри процесса.

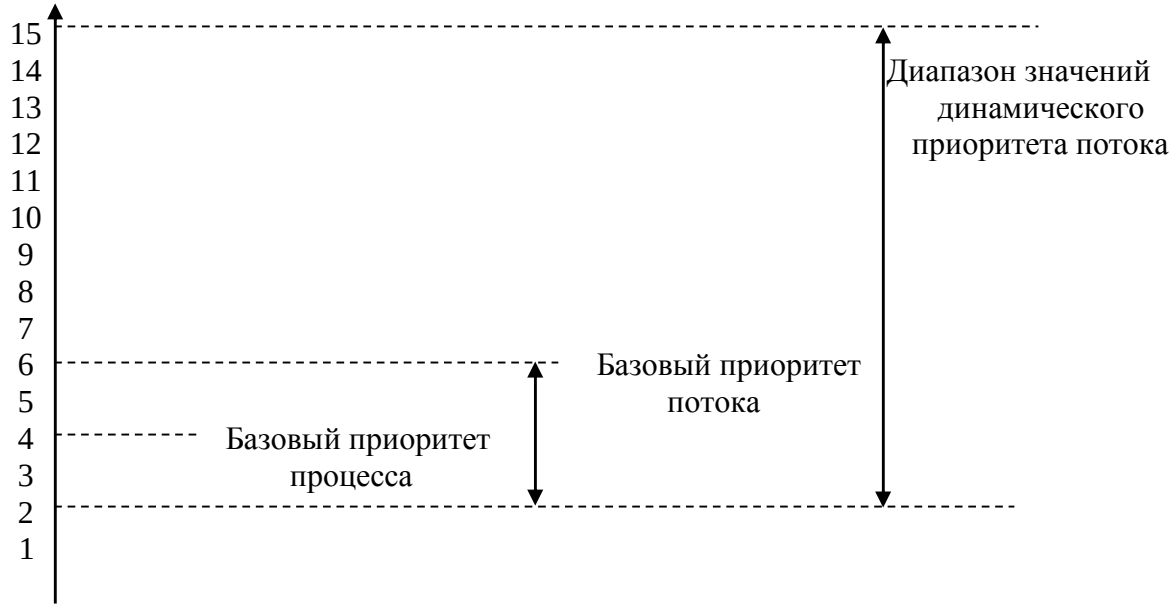


Рис.5. Схема динамического изменения приоритетов в *Windows NT*

Для определения порядка выполнения потоков диспетчер использует систему приоритетов, направляя на выполнение потоки с высоким приоритетом раньше потоков с низкими приоритетами. Система прекращает исполнение или вытесняет (*preempts*) текущий поток, если становится готовой к выполнению другая задача (поток) с более высоким приоритетом.

Существует группа очередей – по одной для каждого приоритета. *Windows NT* поддерживает 32 уровня приоритетов; потоки делятся на два класса: реального времени и переменного приоритета. Потоки реального времени, имеющие приоритеты от 16 до 31 – это высокоприоритетные потоки, используемыми программами с критическим временем выполнения, то есть требующие немедленного внимания системы (по терминологии *Microsoft*).

Диспетчер задач просматривает очереди, начиная с самой приоритетной. При этом если очередь пустая, то есть нет готовых к выполнению задач с таким приоритетом, осуществляется переход к следующей очереди. Следовательно, если есть задачи, требующие процессор немедленно, они будут обслужены в

первую очередь. Для собственно системных модулей, функционирующих в статусе задач, зарезервирована очередь с номером 0.

Большинство потоков в системе относятся к классу переменного приоритета с уровнями приоритета (номером очереди) от 1 до 15. Эти очереди используются потоками с переменным приоритетом (variable priority), так как диспетчер задач корректирует их приоритеты по мере выполнения задач для оптимизации отклика системы. Диспетчер приостанавливает исполнение текущего потока после того, как тот израсходует свой квант времени. При этом если прерванный тред – это поток переменного приоритета, то диспетчер задач понижает его приоритет на единицу и перемещает в другую очередь. Таким образом, приоритет потока, выполняющего много вычислений, постепенно понижается (до значения его базового приоритета). С другой стороны, диспетчер повышает приоритет потока после освобождения задачи (потока) из состояния ожидания. Обычно добавка к приоритету потока определяется кодом исполнительной системы, находящимся вне ядра ОС, однако величина этой добавки зависит от типа события, которого ожидал заблокированный тред. Так, например, поток, ожидавший ввода очередного байта с клавиатуры, получает большую добавку к значению своего приоритета, чем процесс ввода/вывода, работавший с дисковым накопителем. Однако в любом случае значение приоритета не может достигнуть 16.

Управление памятью

Основная память (*memory*) представляет собой упорядоченный массив однобайтовых ячеек, каждая из которых имеет свой уникальный адрес (номер). Процессор извлекает команду из основной памяти, декодирует и выполняет ее. Для выполнения команды могут потребоваться обращения еще к нескольким ячейкам основной памяти. Обычно основная память изготавливается с применением полупроводниковых технологий и теряет свое содержимое при отключении питания.

Вторичную память (*storage*) также можно рассматривать как одномерное линейное адресное пространство, состоящее из последовательности байтов. Как правило, внешняя память реализована с использованием различного рода дисков. В отличие от ОП, она является энергонезависимой, имеет существенно большую емкость и используется в качестве расширения основной памяти.

Эту схему можно дополнить еще несколькими промежуточными уровнями. Разновидности памяти могут быть объединены в иерархию по *убыванию времени доступа, возрастанию цены и увеличению емкости* (рис. 1).

Управление памятью, наряду с управлением процессами и ресурсами, – одна из наиболее важных функций операционной системы. Задача ОС заключается в том, чтобы размещать в памяти пользовательские процессы, их данные, обслуживать запросы процессов на области памяти заданных размеров.

В идеале программисты хотели бы иметь неограниченную в размере и скорости память, которая была бы энергонезависимой, т.е. сохраняла свое содержимое при выключении электричества, а также недорого бы стоила. Однако реально пока такой памяти нет. Чтобы найти выход из сложившийся ситуации, необходимо опираться не на отдельно взятые компоненты или технологию, а выстроить иерархию запоминающих устройств, показанную на рис. 1. При перемещении слева направо происходит следующее:

- снижается стоимость бита;
- возрастает емкость;
- возрастает время доступа;
- снижается частота обращений процессора к памяти.

Кэш-диска – особый тип оперативной памяти, который выступает в роли буфера для хранения промежуточных данных, которые уже считаны с жесткого диска, но еще не были переданы для дальнейшей обработки, а также для хранения данных, к которым система обращается довольно часто. Необходимость наличия транзитного хранилища вызвана разницей между скоростью считывания данных с жесткого диска и пропускной способностью системы.

Кэш центрального процессора разделён на несколько уровней. Для универсальных процессоров — до 3. Кэш-память уровня N+1 как правило больше по размеру и медленнее по скорости обращения и передаче данных, чем кэш-память уровня N.

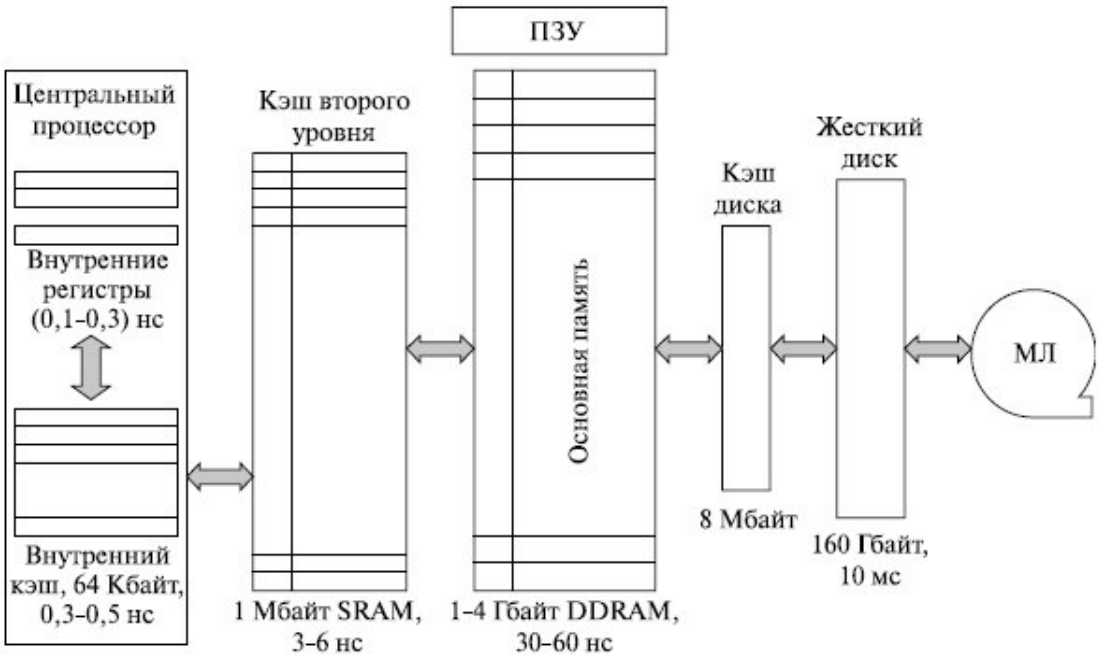


Рис. 1. Иерархия памяти

Виртуальное и физическое адресное пространство

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются символьные имена, виртуальные (математические, условные, логические – все это синонимы) и физические адреса (рис. 2).

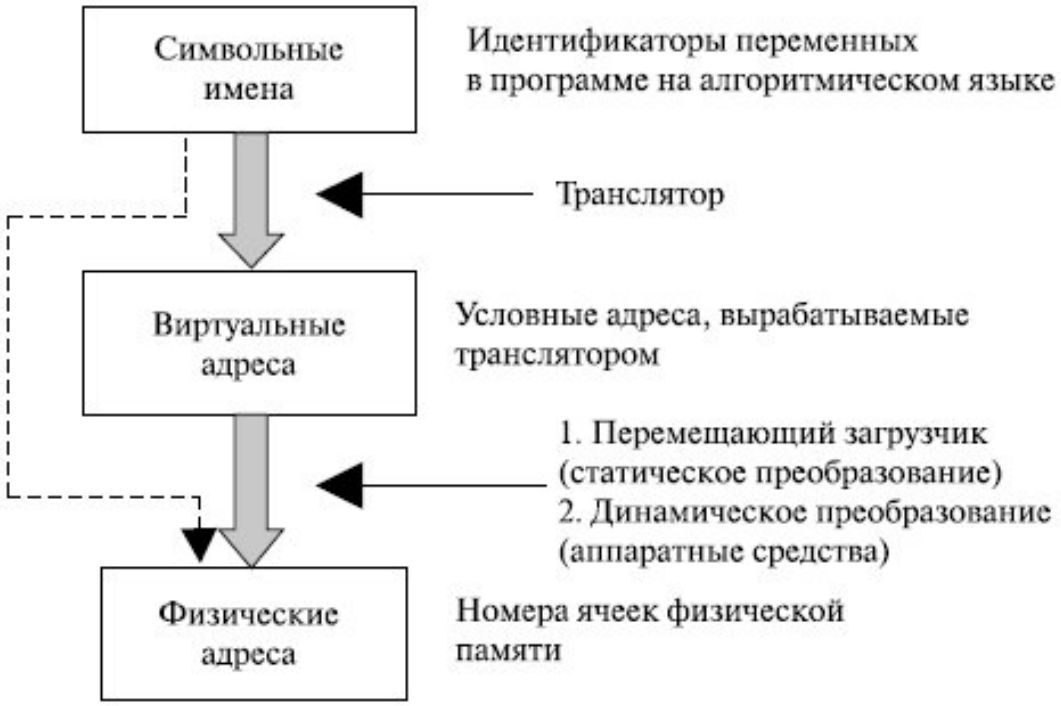


Рис. 2. Типы адресов

Символьные имена присваивает пользователь при написании программ на алгоритмическом языке или ассемблере. Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности будут расположены переменные и команды, их "видит" и "понимает" *устройство управления памятью (Memory Management Unit – MMU).*

Логические адреса совпадают с физическими при связывании адресов во время компиляции или во время загрузки (т.е. *до* исполнения программы). Однако при связывании адресов во время выполнения логические адреса отличаются от физических.

Совокупность виртуальных адресов процесса называется виртуальным адресным пространством. Диапазон адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для 32- разрядного процессора *Pentium* адресное пространство составляет объем, равный 2^{32} байт, с диапазоном адресов от 0000.0000_{16} до $FFFF.FFFF_{16}$).

Устройство управления памятью

Когда используется виртуальная память, виртуальные адреса не передаются напрямую шиной памяти. Вместо этого они передаются диспетчеру памяти (*MMU – Memory Management Unit*), который отображает виртуальные адреса на физические адреса памяти, как показано на рис. 3. Здесь диспетчер памяти показан как часть микросхемы процессора, как обычно и бывает чаще всего. Но логически он мог бы быть отдельной микросхемой, как было в недавнем прошлом.

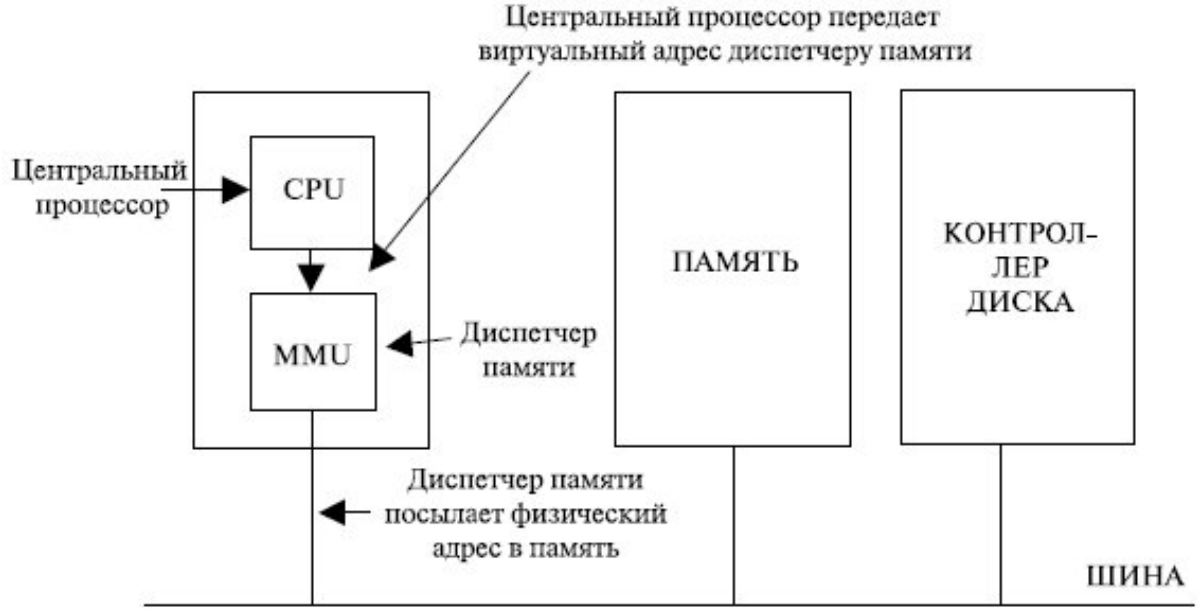


Рис. 3. Диспетчер памяти

Аппаратура *MMU* использует значение *регистра перемещения*, содержащего адрес начала области памяти, выделенной ОС для программы пользователя. *MMU* добавляет значение регистра перемещения к (логическому) адресу, сгенерированному пользовательской программой, получая в результате физический адрес.

Программа пользователя работает только с логическими адресами и не "видит" физических адресов.

Функции ОС по управлению памятью

Под памятью здесь подразумевается оперативная память компьютера Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Особая роль памяти объясняется тем, что процессор может выполнять инструкции программы только в том случае, если они находятся в памяти. Память распределяется как между модулями прикладных программ, так и между модулями самой операционной системы.

Эффективное управление памятью жизненно важно для многозадачных систем. Если в памяти будет находиться небольшое число процессов, то значительную часть времени процессы будут находиться в состоянии ожидания ввода-вывода и загрузка процессора будет низкой.

В ранних ОС управление памятью сводилось просто к загрузке программы и ее данных из некоторого внешнего накопителя в память. С появлением мультипрограммирования перед ОС были поставлены новые задачи, связанные с распределением имеющейся памяти между несколькими одновременно выполняющимися программами.

Функциями ОС по управлению памятью в мультипрограммных системах являются:

- отслеживание (учет) свободной и занятой памяти;
- первоначальное и динамическое выделение памяти процессам приложений и самой операционной системе и освобождение памяти по завершении процессов;
- настройка адресов программы на конкретную область физической памяти;
- полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП недостаточны для размещения всех процессов, и возвращение их в ОП;
- защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;
- дефрагментация памяти.

Алгоритмы распределения памяти

Существует ряд базовых вопросов управления памятью, которые в различных ОС решаются по-разному.

Ниже приводится классификация методов распределения памяти, в которой выделено два класса методов – с перемещением сегментов процессов между ОП и ВП (диском) и без перемещения, т.е. без привлечения внешней памяти (рис. 4). Данная классификация учитывает только основные признаки методов. Для каждого метода может быть использовано несколько различных алгоритмов его реализации.

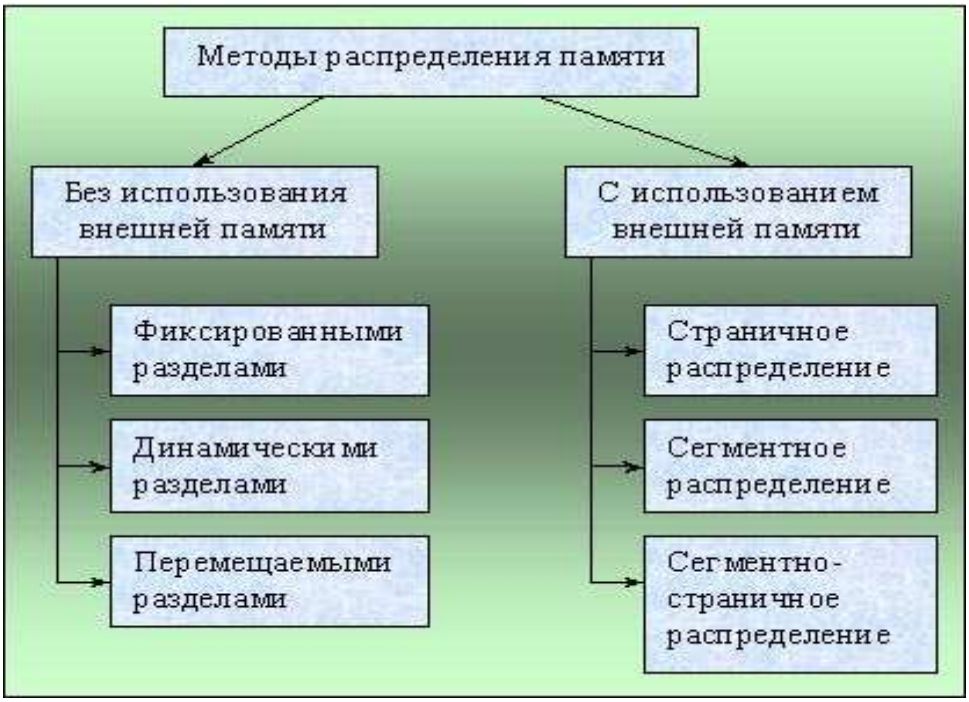


Рис. 4. Классификация методов распределения памяти

Распределение фиксированными разделами

Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых *разделами*. Границы разделов не изменяются.

Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь либо в очередь к некоторому разделу.

Подсистема управления памятью в этом случае выполняет следующие задачи.

Сравнивает объем памяти, требуемый для вновь поступившего процесса, с размерами свободных разделов и выбирает подходящий раздел. Осуществляет загрузку программы в один из разделов и настройку адресов.

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

Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени.

Распределение памяти динамическими разделами

В этом случае память машины не делится заранее на разделы. Сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

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

- Ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти.
- При создании нового процесса – анализ требований к памяти, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения кодов и данных нового процесса. Выбор раздела может осуществляться по разным правилам, например: "первый попавшийся раздел достаточного размера", "раздел, имеющий наименьший достаточный размер" или "раздел, имеющий наибольший достаточный размер".
- Загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей. Данный способ предполагает, что программный код не перемещается во время выполнения, а значит, настройка адресов может быть проведена единовременно во время загрузки.
- После завершения процесса корректировка таблиц свободных и занятых областей.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – *фрагментация* памяти. Фрагментация – то наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов).

Перемещаемые разделы

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

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

Концепция сжатия применяется и при использовании других методов распределения памяти, когда отдельному процессу выделяется не одна сплошная область памяти, а несколько несмежных участков памяти произвольного размера (сегментов).

Свопинг и виртуальная память

Необходимым условием для того, чтобы программа могла выполняться, является ее нахождение в оперативной памяти. Только в этом случае процессор может извлекать команды из памяти и интерпретировать их, выполняя заданные действия. В условиях, когда для обеспечения приемлемого уровня мультипрограммирования имеющейся оперативной памяти недостаточно, был предложен метод организации вычислительного процесса, при котором образы некоторых неактивных процессов целиком или частично временно выгружаются на диск. К моменту, когда подходит очередь выполнения выгруженного процесса, его образ возвращается с диска в оперативную память.

Такая подмена *(виртуализация)* оперативной памяти дисковой памятью позволяет повысить уровень мультипрограммирования – объем оперативной памяти компьютера теперь не столь жестко ограничивает количество одновременно выполняемых процессов.

Виртуализация памяти может быть осуществлена на основе двух различных подходов:

- *свопинг (swapping – подкачка и откачка)* – образы процессов выгружаются на диск и возвращаются в оперативную память целиком;
- *виртуальная память (virtual memory)* – между оперативной памятью и диском перемещаются части (сегменты, страницы и т. п.) образов процессов.

Свопинг представляет собой частный случай виртуальной памяти и, следовательно, более простой в реализации. Однако подкачке свойственна избыточность: когда ОС решает активизировать процесс, для его выполнения, как правило, не требуется загружать в оперативную память все его сегменты

полностью. Перемещение избыточной информации замедляет работу системы, а также приводит к неэффективному использованию памяти.

Именно из-за указанных недостатков свопинг как основной механизм управления памятью почти не используется в современных ОС. На смену ему пришел более совершенный механизм виртуальной памяти.

Ключевой проблемой виртуальной памяти, возникающей в результате многократного изменения местоположения в оперативной памяти образов процессов или их частей, является преобразование виртуальных адресов в физические. Решение этой проблемы зависит от того, какой способ структуризации виртуального адресного пространства принят в данной системе управления памятью. В настоящее время все множество реализаций виртуальной памяти может быть представлено тремя классами.

1. *Страничная виртуальная память* организует перемещение данных между памятью и диском страницами – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера.

Страничная организация памяти (*paging*) – наиболее распространенная стратегия управления памятью, используемая практически во всех операционных системах. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами (*virtual pages*). Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов.

2. *Сегментная виртуальная память* предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных.

Сегментная организация памяти (*segmentation*) – схема распределения памяти в виде сегментов переменной длины, соответствующая пользовательской трактовке распределения памяти, т.е. логической структуре программ и данных. Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями.

3. *Сегментно-страничная* виртуальная память использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных здесь является страница.

Данный метод представляет собой комбинацию страничного и сегментного механизмов управления памятью и направлен на реализацию достоинств обоих подходов.

Кэш-память

Кэш-память – это способ совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который за счет динамического копирования в "быстрое" ЗУ наиболее часто используемой информации из "медленного" ЗУ позволяет, с одной стороны, уменьшить среднее время доступа к данным, а с другой стороны, экономить более дорогую быстродействующую память.
Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств - "быстрое" ЗУ.

В процессе работы содержимое кэш-памяти постоянно обновляется, а значит, время от времени данные из нее должны вытесняться. Алгоритм замены данных в кэш-памяти существенно влияет на ее эффективность. В идеале такой алгоритм должен, во-первых, быть максимально быстрым, чтобы не замедлять работу кэш-памяти, а во-вторых, обеспечивать максимально возможную вероятность кэш-попаданий.

При *кэшировании* данных из оперативной памяти широко используются две основные схемы отображения: *случайное отображение* и *детерминированное отображение*.

При *случайном* отображении элемент оперативной памяти в общем случае может быть размещен в произвольном месте кэш-памяти. Для того чтобы в дальнейшем можно было найти нужные данные в кэше, они помещаются туда вместе со своим адресом. При каждом запросе к оперативной памяти выполняется поиск в кэше, причем критерием поиска выступает адрес оперативной памяти из запроса.

Второй, *детерминированный способ* отображения предполагает, что любой элемент основной памяти всегда отображается в одно и то же место кэш-памяти. В этом случае кэш-память разделена на строки, каждая из которых предназначена для хранения одной записи об одном элементе данных и имеет свой номер.

Управление параллельными взаимодействующими вычислительными процессами

Основной особенностью мультипрограммных ОС является то, что в их среде параллельно развивается несколько вычислительных процессов. *Параллельными* называют последовательные вычислительные процессы, которые одновременно находятся в каком-либо активном состоянии. Два параллельных процесса могут быть независимыми либо взаимодействующими.

Независимыми являются процессы, множества переменных которых не пересекаются. Под переменными в этом случае понимают файлы данных, а также области оперативной памяти, сопоставленные определенным в программе (и промежуточным) переменным. Независимые процессы не влияют на работу друг друга. Они могут только быть причиной задержек исполнения других процессов, так как вынуждены разделять ресурсы системы.

Взаимодействующие процессы совместно используют общие переменные, и выполнение одного процесса может повлиять на выполнение другого.

Взаимодействовать могут либо *конкурирующие* процессы, либо процессы, совместно выполняющие общую работу.

Процессы, выполняющие общую совместную работу таким образом, что результаты вычислений одного процесса в явном виде передаются другому (обмен данными), называются *сотрудничающими*.

Чтобы предотвратить некорректное исполнение конкурирующих процессов вследствие нерегламентированного доступа к разделяемым (общим) переменным, необходимо ввести механизм *взаимного исключения*, который не позволит двум процессам одновременно обращаться к разделяемым переменным. Такие общие переменные называют *критическими ресурсами*.

Кроме этого, в ОС должны быть предусмотрены средства, синхронизирующие работу взаимодействующих процессов. Другими словами, процессы должны обращаться к неким средствам не только ради синхронизации с целью взаимного исключения, но и чтобы обмениваться данными.

Те места в программах, где происходит обращение к критически ресурсам, называются *критическими секциями* или *критическими интервалами*. Необходима организация такого доступа к критическому ресурсу, когда только одному процессу разрешается входить в критическую секцию.

Обеспечение взаимоисключения является одной из ключевых проблем параллельного программирования. Было предложено несколько способов решения этой проблемы – программные и аппаратные.

Использование блокировки памяти при синхронизации параллельных процессов

Все вычислительные системы имеют такое средство для организации взаимного исключения, как *блокировка памяти*. Это средство запрещает

одновременное использование двух (и более) команд, которые обращаются к одной и той же ячейке памяти. Механизм блокировки памяти предотвращает одновременный доступ к разделяемой переменной, но не предотвращает чередование доступа. Таким образом, если критические интервалы исчерпываются одной командой обращения к памяти, данного средства может быть достаточно для непосредственной реализации взаимного исключения. Если критические секции требую более одного обращения к памяти, задача становится сложной, но алгоритмически разрешимой.

Рассмотрим следующую модель двух взаимодействующих процессов (рис.1).

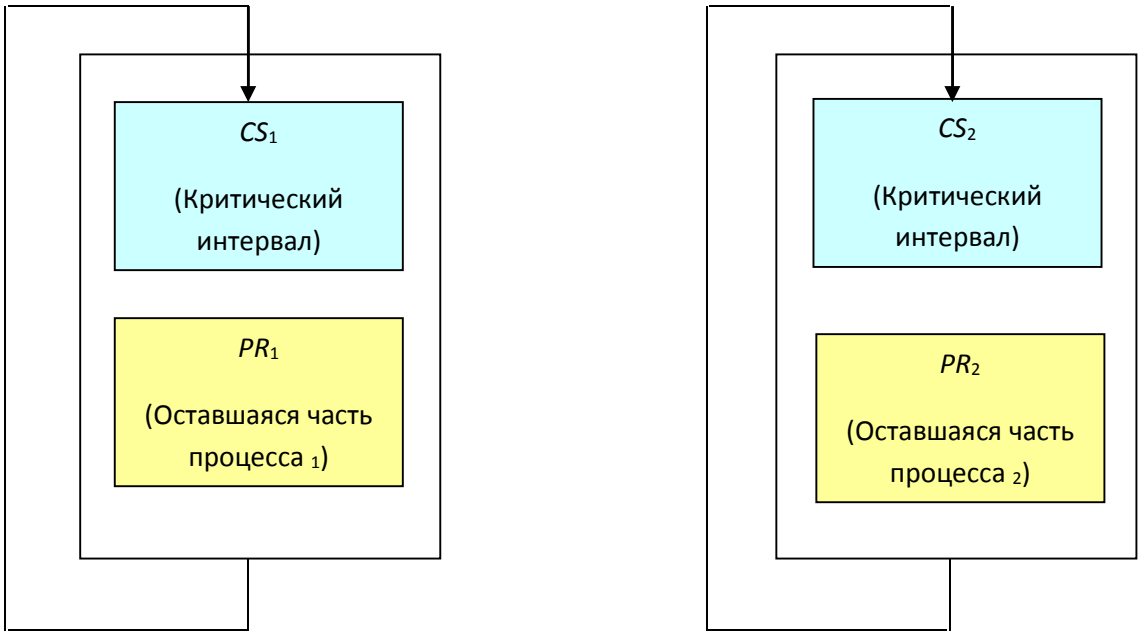


Рис. 1. Модель взаимодействующих процессов

Пусть имеется два циклических процесса, в которых есть критические секции, т.е. каждый из процессов состоит из двух частей:

- некоторого критического интервала;
- оставшаяся часть кода, в которой нет работы с общими переменными.

Пусть эти два процесса асинхронно разделяют по времени единственный процессор, либо выполняются на отдельных процессорах, каждый из которых имеет доступ к некоторой общей части памяти, с которой и работают критические секции.

Рассмотрим вариант решения взаимного исключения, использующий только блокировку памяти. Это известный алгоритм, предложенный математиком Деккером. *Алгоритм Деккера* основан на использовании 3-х переменных: *перекл*1, *перекл*2 и *ОЧЕРЕДЬ*. С каждым из процессов CS_1 и CS_2 будут связаны соответственно переменные *перекл*1 и перекл2, по смыслу являющиеся переключателями, которые принимают значение true, когда

соответствующий процесс находится в своем критическом интервале, и false – в противном случае. Переменная целого типа *ОЧЕРЕДЬ* указывает, чье сейчас право сделать попытку входа, при условии, что оба процесса хотят выполнить свои критические интервалы.

Если *перекл*2= true и перекл1= false, то выполняется критический интервал для процесса CS_2 независимо от значения переменной *ОЧЕРЕДЬ*. Аналогично для случая *перекл*1= true и *перекл*2= false. Если же оба процесса хотят выполнить свои критические интервалы, т.е. *перекл*2= true и *перекл*1= true, то выполняется критический интервал того процесса, на который указывало значение переменной *ОЧЕРЕДЬ*, независимо от скоростей развития процесса.

Синхронизация процессов посредством операции «ПРОВЕРКА И УСТАНОВКА»

Операция «ПРОВЕРКА И УСТАНОВКА» является, как и блокировка памяти, одним из аппаратных средств решения задачи критического интервала. Данная операция реализована во многих компьютерах. Так в IBM 360 эта команда называлась *TS* (test and set). Действие этой двухоперандной команды заключается в том, что процессор присваивает значение второго операнда первому, после чего второму операнду присваивается значение, равное единице. Эта команда является неделимой, то есть между ее началом и концом не могут выполняться никакие другие команды.

Чтобы использовать команду *TS* для решения проблемы критического интервала, свяжем с ней переменную *common*, которая будет общей для всех процессов, использующих некоторый критический ресурс. Данная переменная будет принимать единичное значение, если какой-либо из взаимодействующих процессов находится в своем критическом интервале. С каждым процессом связана своя локальная переменная *local*, которая принимает значение, равное единице, если данный процесс хочет войти в свой критический интервал. Операция *TS* будет присваивать значение *common* переменной *local* и устанавливать *common* в единицу.

Пусть значение *common* равно нулю. Предположим, что первым захочет войти в свой критический интервал процесс CS_1. В этом случае значение *local*1 установится в единицу, а после цикла проверки с помощью команды *TS* (*local*1, *common*) – в ноль. При этом значение *common* станет равным единице. Процесс CS_1 войдет в свой критический интервал. После выполнения этого критического интервала *common* примет значение равное нулю, что даст возможность второму процессу CS_2 войти в свой критический интервал.

В микропроцессорах *i*80386 и старше есть специальные команды *BTC, BTS* (*bit test and reset* – проверка бита и установка), *BTR*, которые как раз и являются вариантами реализации команды типа «ПРОВЕРКА И УСТАНОВКА».

Несмотря на то, что и алгоритм Деккера и операция «ПРОВЕРКА И УСТАНОВКА» пригодны для реализации взаимного исключения, оба эти приема очень неэффективны. Всякий раз, когда один из процессов выполняет свой критический интервал, любой другой процесс, который пытается войти в свою критическую секцию, попадает в цикл проверки соответствующих переменных-флагов, регламентирующих доступ к критическим переменным (находится в *активном ожидании*). В результате имеем общее замедление вычислительной системы процессами, которые реально не выполняют никакой полезной работы.

До тех пор, пока процесс, занимающий в данный момент критический ресурс, не решит его уступить, все другие процессы, ожидающие этого ресурса, могли бы вообще не конкурировать за процессорное время. Для этого их надо перевести в состояние ожидания (заблокировать их выполнение)

Вместо того чтобы связывать с каждым процессом свою собственную переменную можно со всем множеством конкурирующих критических секций связать одну переменную, которую и рассматривать как некоторый «ключ». Перед входом в свой критический интервал процесс забирает «ключ» и тем самым блокирует другие процессы. Таким образом, каждый процесс, входящий в критический интервал, должен вначале проверить, доступен ли «ключ», и если это так, то сделать его недоступным для других процессов. Самым главным здесь является то, что эти два действия должны быть неделимыми, чтобы два или более процессов не могли одновременно получить доступ к «ключу».

Таким образом, мы подошли к одному из главных механизмов решения проблемы взаимного исключения – семафорам Дейкстры.

Семафорные примитивы Дейкстры

Семафор – переменная специального типа, которая доступна параллельным процессам для проведения над ней только двух операций: «закрытии» и «открытия», названных соответственно *P*- и *V*-операциями. Эти операции являются примитивами относительно семафора, который указывается в качестве параметра операции. Здесь семафор выполняет роль вспомогательного критического ресурса, так как операции *P* и *V* неделимы при своем выполнении и взаимно исключают друг друга.

Основным достоинством семафорных операций является отсутствие состояния «активного ожидания», что повышает эффективность работы мультипрограммной вычислительной системы.

В настоящее время используют много различных семафорных механизмов. Варьируемыми параметрами, которые отличают различные виды примитивов, являются:

- начальное значение и диапазон изменения значений семафора;
- логика действий семафорных операций;

• количество семафоров, доступных для обработки при исполнении отдельного примитива.

Обобщенный смысл примитива *P*(*S*) состоит в проверке текущего значения семафора *S* (*P* – от голландского ***P****roberen* – проверить), и если оно не меньше нуля, то осуществляется переход к следующей за примитивом операции. В противном случае процесс снимается на некоторое время с выполнения и переводится в состояние «пассивного ожидания». Находясь в списке заблокированных, ожидающий процесс не проверяет семафор непрерывно, как в случае активного ожидания.

Операция *V*(*S*) связана с увеличением значения семафора на единицу (*V* – от голландского ***V****erhogen* – увеличить) и переводом одного или нескольких процессов в состояние готовности к центральному процессору.

Операции *P* и *V* выполняются операционной системой в ответ на запрос, выданный некоторым процессом и содержащий имя семафора в качестве параметра.

Допустимыми значениями семафора являются только целые числа. *Двоичным семафором* называют семафор, максимальное значение которого равно единице. В противном случае семафоры называют *N*-ичными.

Рассмотрим на нашем примере двух конкурирующих процессов использование данных семафорных примитивов для решения проблемы критического интервала:

1) семафор имеет начальное значение, равное 1. Если процессы CS_1 и CS_2 попытаются одновременно выполнить примитив *P*(*S*), то это удастся сделать только одному из них. Пусть это будет CS_2:

2) процесс CS_2 закрывает семафор *S* (выполняет операцию *P*(*S*)) и выполняет свой критический интервал. Процесс CS_1 будет заблокирован на семафоре *S*. Тем самым гарантируется взаимное исключение;

3) после выполнения примитива *V*(*S*) процессом CS_2 семафор *S* открывается, указывая на возможность захвата каким-либо процессом освободившегося критического ресурса. При этом производится перевод процесса CS_1 из заблокированного состояния в состояние готовности.

Мониторы Хоара

Несмотря на очевидные достоинства (простота, независимость от количества процессов, отсутствие «активного ожидания») семафорные механизмы имеют и ряд недостатков. Они слишком примитивны, так как семафор не указывает непосредственно на синхронизирующее условие, с которым он связан, или на критический ресурс. Поэтому при построении сложных схем синхронизации алгоритмы получаются весьма непростыми и ненаглядными.

Необходимо иметь понятные, очевидные решения, которые позволят программистам создавать параллельные взаимодействующие программы. К таки решениям можно отнести так называемые *мониторы*, предложенные Хоаром.

В параллельном программировании *монитор* – это пассивный набор разделяемых переменных и повторно входимых процедур доступа к ним, которым процессы пользуются в режиме разделения, причем в каждый момент им может пользоваться только один процесс.

Монитор – это механизм организации параллелизма, который содержит как данные, так и процедуры, необходимые для реализации динамического распределения общего ресурса. Процесс, желающий получить доступ к разделяемым переменным, должен обратиться к монитору, который либо предоставит доступ, либо откажет в нем. Вход в монитор находится под жестким контролем – здесь осуществляется взаимоисключение процессов, так что в каждый момент времени только одному процессу можно войти в монитор. Процессам, которым надо войти в монитор, когда он уже занят, приходится ждать, причем режимом ожидания автоматически управляет сам монитор. При отказе в доступе монитор блокирует обратившийся к нему процесс и определяет условие, по которому процесс ждет. Внутренние данные монитора могут быть либо глобальными (относящимися ко всем процедурам монитора), либо локальными (относящимися только к одной конкретной процедуре). Ко всем этим данным можно обращаться только изнутри монитора. При первом обращении монитор присваивает своим переменным начальные значения. При каждом новом обращении используются те значения переменных, которые сохранились от предыдущего обращения.

Со временем процесс, который занимал данный ресурс, обратится к монитору, чтобы возвратить ресурс системе. Чтобы гарантировать, что процесс, находящийся в ожидании некоторого ресурса, со временем получит этот ресурс, делается так, что ожидающий процесс имеет более высокий приоритет, чем новый процесс, пытающийся войти в монитор.

По сравнению с семафорами мониторы обеспечивают значительное упрощение организации взаимодействующих вычислительных процессов и большую наглядность лишь при незначительной потере в эффективности.

Почтовые ящики

Тесное взаимодействие между процессами предполагает не только синхронизацию – обмен временными сигналами, но и передачу, и получение произвольных данных – обмен сообщениями. В системе с одним процессором посылающий и принимающий процессы не могут работать одновременно. Следовательно, для хранения посланного, но еще не полученного сообщения необходимо место. Оно называется *буфером сообщений* или *почтовым ящиком*. Это информационная структура, поддерживаемая операционной системой. Она состоит из головного элемента, в котором находится информация о данном

почтовом ящике, и из нескольких буферов (гнезд), в которые помещают сообщения. Размер каждого буфера, их количество обычно задаются при образовании почтового ящика.

Основные достоинства почтовых ящиков:

- процессу не нужно знать о существовании других процессов до тех пор, пока он не получит сообщения от них;
- два процесса могут обмениваться более чем одним сообщением за один раз;
- операционная система может гарантировать, что никакой процесс не вмешается в «беседу» других процессов:
- очереди буферов позволяют процессу-отправителю продолжать работу, обращая внимания на получателя.

Основным недостатком буферизации сообщений является появление еще одного ресурса, которым нужно управлять.

Конвейеры (программные каналы)

Конвейер (*pipe* –программный канал, *транспортер*) является средством, с помощью которого можно производить обмен данными между процессами. Принцип работы конвейера основан на механизме ввода/вывода, который используется с файлами в *UNIX* – задача, передающая информацию, действует так, как будто она записывает данные в файл, в то время как задача, для которой предназначается эта информация, читает ее из этого файла. Операции записи и чтения осуществляются не записями, как это делается в обычных файлах, а потоком байтов, как это делается в *UNIX*-системах. Функции, с помощью которых выполняется запись в канал и чтение из него, являются теми же самыми, что и при работе с файлами. На самом деле конвейеры являются не файлами на диске, а представляют собой буферную память, работающую по принципу обычной очереди (*FIFO – first input, first output*)

Конвейер имеет максимальный размер 64 Кбайт и работает циклически.

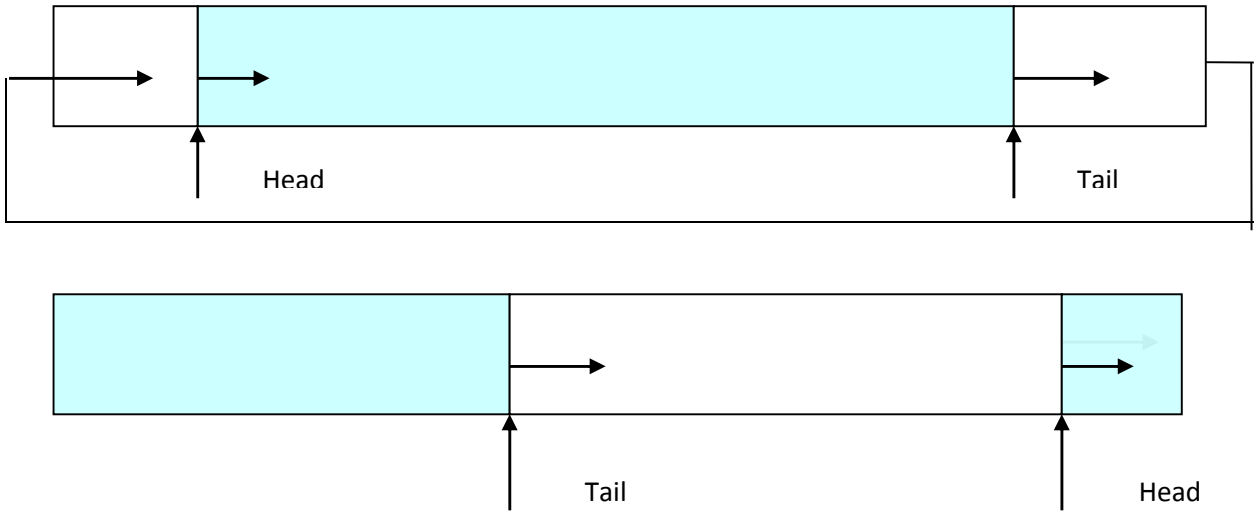


Рис. 2. Организация очереди на массиве

Имеется некий массив и два указателя, равные нулю в начальный момент:

- один показывает на первый элемент – *Head* (Голова);
- второй – на последний – *Tail* (Хвост).

Запись (добавление) одного элемента в пустую очередь делает оба указателя, равными единице. В последующем при добавлении нового элемента увеличивается указатель *Tail*. Чтение (удаление) элемента производится, начиная с первого элемента созданной очереди. При этом, следовательно, будет увеличиваться указатель *Head*. Таким образом, в результате операций записи и чтения указатели будут перемещаться от начала массива к его концу. При достижении указателем значения индекса последнего элемента, значение указателя вновь становится равным единице. Массив как бы замыкается в кольцо, организуя круговое перемещение указателей. Сказанное проиллюстрировано на рис. 2.

Конвейеры представляют системный ресурс. Чтобы начать работу с конвейером, процесс должен заказать его у операционной системы и получить в свое распоряжение. Процессы, знающие идентификатор конвейера, могут через него обмениваться данными.

5. Очереди сообщений

Очереди сообщений (*queue*) являются более сложным методом связи между взаимодействующими процессами по сравнению с каналами. С помощью очередей также можно из одной или нескольких задач независимым образом посылать сообщения некоторой задаче-приемнику. При этом только процесс-приемник может читать и удалять сообщения из очереди, а процессы-клиенты имеют право лишь помещать в очередь свои сообщения. Таким образом, очередь

работает в одном направлении. Если необходима двухсторонняя связь, то можно создать две очереди.

Отличия работы с очередями от работы с конвейерами:

- очереди предоставляют возможность использовать несколько дисциплин обработки сообщений:
  - *FIFO* – сообщение, записанное первым, будет первым и прочитано;
  - *LIFO* – сообщение, записанное последним, будет прочитано первым:
  - приоритетный – сообщения читаются с учетом их приоритета;
  - произвольный доступ, т.е. можно читать любое сообщение, тогда как конвейер обеспечивает только дисциплину *FIFO*.
- при чтении сообщения из конвейера оно из него удаляется. При чтении из очереди этого не происходит, сообщение при желании может быть прочитано несколько раз;
- в очередях присутствуют не непосредственно сами сообщения, а только их адреса в памяти и размер.

Проблема тупиков и методы борьбы с ними

При организации параллельного выполнения нескольких процессов одной из главных функций ОС является решение сложной задачи корректного распределения ресурсов между выполняющимися процессами и обеспечение последних средствами взаимной синхронизации и обмена данными.

При параллельном исполнении процессов могут возникать ситуации, при которых два или более процессов все время находятся в заблокированном состоянии. Самым простым является случай, когда каждый из двух процессов ожидает ресурс, занятый другим процессом. Эта тупиковая ситуация называется *дедлоком* (*dead lock* –смертельное объятие), тупиком или клинчем. Тупики чаще всего возникают из-за конкуренции несвязанных параллельных процессов за ресурсы вычислительной системы, но иногда к тупикам приводят и ошибки программирования.

Рассмотрим пример, поясняющий причину возникновения тупиков.

Пусть имеются три процесса ПР1, ПР2, ПР3, которые вырабатывают соответственно сообщения М1, М2, М3. Пусть процесс ПР1 является «потребителем» сообщения М3, процесс ПР2 получает сообщение М1, а ПР3 – сообщение М2 от процесса ПР2, то есть каждый из процессов является и «поставщиком» и «потребителем» одновременно и вместе они образуют «кольцевую» систему передачи сообщений через почтовые ящики (ПЯ) – рис.3.

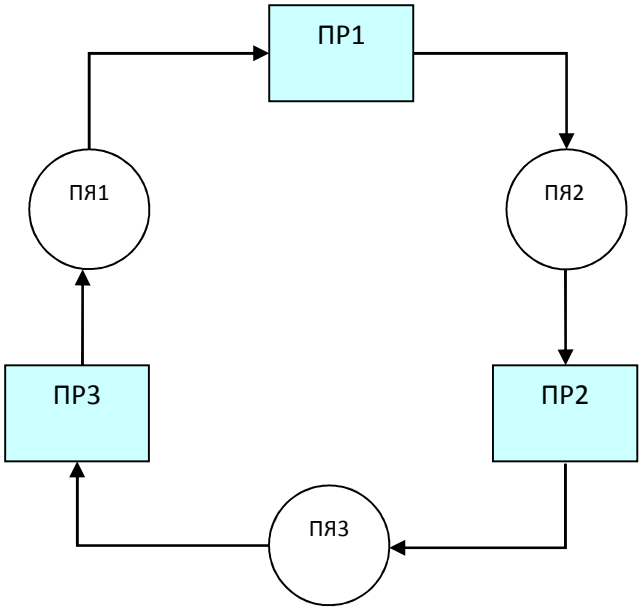


Рис. 3. Кольцевая схема взаимодействия процессов

Если процедуры в каждом процессе выполняются в следующем порядке:

ПР1: …
Послать сообщение (ПР2, М1, ПЯ2);
Ждать сообщение (ПР3, М3, ПЯ1);
…
ПР2: …
Послать сообщение (ПР3, М2, ПЯ3);
Ждать сообщение (ПР1, М1, ПЯ2);
…
ПР3: …
Послать сообщение (ПР1, М3, ПЯ1);
Ждать сообщение (ПР2, М2, ПЯ3);
…,

то никаких трудностей не возникает. Однако перестановка этих двух процедур в каждом из процессов (Ждать…; Послать…;) вызывает тупик. В самом деле, в этом случае ни один из процессов не может послать сообщения до тех пор, пока сам его не получит, а этого события никогда не произойдет, поскольку ни один процесс не может этого сделать.

Коффман и другие исследователи доказали, что для возникновения тупиковой ситуации должны выполняться четыре условия.

1. *Условие взаимного исключения*.

Каждый ресурс в данный момент или отдан ровно одному процессу, или доступен.

2. *Условие удерживания и ожидания*.

Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.

3. *Условие отсутствия принудительной выгрузки ресурсов*.

У процесса нельзя забрать принудительно ранее полученные ресурсы. Процесс, владеющий ими, должен сам освободить ресурсы.

4. *Условие циклического ожидания*.

Должна существовать круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

Для того чтобы возникла тупиковая ситуация (произошла взаимоблокировка), должны выполняться все эти четыре условия. Если хотя бы одно отсутствует, тупиковая ситуация невозможна.

Для борьбы с тупиковыми ситуациями используются четыре стратегии.

1. Пренебрежение проблемой в целом

Если взаимоблокировки случаются в среднем раз в пять лет, а сбои ОС, компиляторов и неисправности аппаратуры происходят в среднем один раз в неделю, то подходит первая стратегия. К этому надо добавить, что большинство операционных систем потенциально страдают от взаимоблокировок, которые не обнаруживаются, не говоря уже об автоматическом выходе из тупика.

2. Предотвращать с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки

Условие взаимного исключения можно подавить путем разрешения неограниченного распределения ресурсов. Это совершенно неприемлемо к совместно используемым переменным в критических интервалах.

Условие удержания и ожидания можно подавить, предварительно выделяя ресурсы. При этом процесс может начать исполнение, только получив все необходимые ресурсы. Это может привести к снижению эффективности работы вычислительной системы в целом. Кроме того, это сделать зачастую невозможно, так как необходимые ресурсы становятся известны процессу только после начала исполнения.

Условие отсутствия принудительной выгрузки ресурсов можно исключить, позволяя ОС отнимать у процесса ресурсы. Перераспределение процессора реализуется достаточно легко, в то время как перераспределение устройств ввода/вывода крайне нежелательно.

Условие циклического ожидания можно исключить, предотвращая образование цепи запросов. Это можно обеспечить с помощью принципа иерархического выделения ресурсов, если порядок использования ресурсов является иерархическим.

В целом стратегия предотвращения тупиков – это очень дорогое решение проблемы, и она используется нечасто.

3. *Обнаружение и восстановление*

При использовании этого метода система не пытается предотвратить попадания в тупиковые ситуации. Вместо этого она позволяет произойти взаимоблокировке, старается определить, когда это случилось, и затем совершает некоторые действия по возврату системы к состоянию, имевшему место до того, как система попала в тупик.

Пусть процесс *A* занимает ресурс *R* и хочет получить ресурс *S*.

Вопрос: заблокирована ли эта система, и если да, то какие процессы в этом участвуют?

Чтобы ответить на этот вопрос, нужно составить граф ресурсов и процессов (рис. 4).

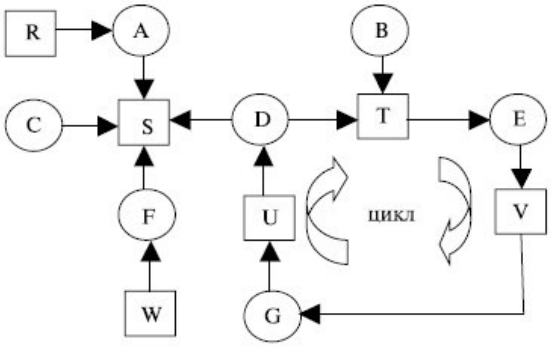


Рис. 4. Граф ресурсов и процессов

Имеем семь процессов *A, B, C. D, E, F, G* и шесть ресурсов *R, S, T, U, V, W*.

Этот граф содержит цикл, указывающий, что процессы *D, E, G* заблокированы (зрительно легко видно). Однако в этом случае в операционной системе необходима реализация формального алгоритма, выявляющего тупики.

Рассмотрим возможность обнаружения взаимоблокировок при наличии нескольких ресурсов каждого типа. Пусть имеется множество процессов $P=\{P_1, P_2,... P_n\}$, всего n процессов, и множество ресурсов $E=\{E_1, E_2,... E_m\}$, где m – число классов ресурсов. В любой момент времени некоторые из ресурсов могут быть заняты и, соответственно, недоступны. Пусть A – вектор доступных ресурсов $A=\{A_1, A2,... A_m\}$. Очевидно, что $A_j \leq E_j$, $j = 1, 2, \ldots, m$.

Введем в рассмотрение две матрицы:

$C=\{c_{i,j} | \ i = 1, 2,\ldots, n; \ j = 1, 2, \ldots, m\}$ – матрица текущего распределения ресурсов, где $c_{i,j}$ – количество ресурсов j-ого класса, которые занимает процесс P_i ;

$R=\{r_{i,j} | \ i = 1, 2,\ldots, n; \ j = 1,2, \ldots, m\}$ – матрица требуемых (запрашиваемых) ресурсов, $r_{i,j}$ – количество ресурсов j-ого класса, которые хочет получить процесс P_i .

Справедливо m соотношений по ресурсам:

$$\sum_{i=1}^{n} Cn_{ij} + A_j = E_j, \ j = 1, 2, \ldots, m.$$

Алгоритм обнаружения взаимоблокировок основан на сравнении векторов доступных и требуемых ресурсов. В исходном состоянии все процессы не маркированы (не отмечены). По мере реализации алгоритма на процессы будет ставиться отметка, служащая признаком того, что они могут закончить свою работу и, следовательно, не находятся в тупике. После завершения алгоритма любой немаркированный процесс находится в тупиковой ситуации.

Алгоритм обнаружения тупиков состоит из следующих шагов.

1. Отыскивается процесс P_i, для которого i-я строка матрицы R меньше вектора A.

2. Если такой процесс найден, это означает, что он может завершиться, а следовательно – освободить занятые ресурсы. Найденный процесс маркируется, и далее прибавляется i-я строка матрицы C к вектору A, т.е. $A_j = A_j + c_{i,j}$, j=1, 2, …, m. Возвращаемся к шагу 1.

3. Если таких процессов не существует, работа алгоритма заканчивается. Немаркированные процессы попадают в тупик.

Рассмотрим методы обнаружения взаимоблокировок.

Обнаружение взаимоблокировки при наличии одного ресурса каждого типа достаточно просто. Для такой системы можно построить граф ресурсов и процессов, о котором уже говорилось, и если в графе нет циклов, система в тупик не попала.

Например, пусть система из семи процессов (*A, B, C, D, E, F, G*) и шести ресурсов (*R, S, T, V,W, U*) в некоторый момент соответствует следующему списку (Рис. 1).

Когда нужно искать возникновение тупиков? Можно, конечно, проверять систему каждый раз, когда запрашивается очередной ресурс, это позволит обнаружить тупик максимально рано, но приведет к большим издержкам процессорного времени. Поэтому период проверки нужно выбрать: например, каждые *K* (сколько – нужно определить!) минут или когда процессор слабо загружен. Стоимость стратегии распознавания тупика зависит от того, насколько часто выполняется алгоритм распознавания. Основная цена восстановления от тупика – это потери времени, которые могут быть существенными.

Предположим, обнаружен тупик. Какие методы можно использовать для его ликвидации? Здесь возможно несколько подходов.

Первый – принудительная выгрузка ресурсов: способность забирать ресурс у процесса, отдавать его другому процессу, а затем возвращать назад так, что исходный процесс не замечает того, в значительной мере зависит от свойств ресурса. Выйти из тупика, таким образом, зачастую трудно или невозможно.

Второй подход – восстановление через откат. В этом способе процессы должны периодически создавать контрольные точки, позволяющие запустить процесс с его предыстории. Когда взаимоблокировка обнаружена, достаточно просто понять, какие ресурсы нужны процессам. Чтобы выйти из тупика,

процесс, занимающий необходимый ресурс, откатывается к тому моменту времени, перед которым он получил данный ресурс, для чего запускается одна из его контрольных точек. Вся работа, выполненная после этой контрольной точки, теряется. Если возобновленный процесс вновь пытается получить данный ресурс, ему придется ждать, когда ресурс станет доступным.

Третий подход – восстановление путем уничтожения одного или более процессов. Это грубейший, но простейший выход из тупика. Проблема – решить, какой процесс уничтожать. Легче всего уничтожать те процессы, которые можно запустить сначала без всяких болезненных эффектов. При этом могут быть уничтожены как процессы, входящие в цикл взаимоблокировки, так и процессы, не находящиеся в цикле, но владеющие ресурсами, необходимыми заблокированным процессам.

4. Избегать тупиковых ситуаций с помощью аккуратного распределения ресурсов

Идеальной была бы такая организация вычислительного процесса, при которой не возникали бы тупики за счет безопасного распределения ресурсов. в большинстве систем процессы запрашиваются не все сразу, а поочередно. Система должна уметь решать, является ли предоставление ресурса безопасным или нет, и предоставить его процессу только в первом случае. Таким образом, возникает вопрос: существует ли алгоритм, который всегда может избежать ситуации взаимоблокировки, все время делая правильный выбор? Ответом будет условие «да» – мы можем избежать тупиков, но, если заранее будет доступна определенная информация.

Основные алгоритмы, позволяющие предотвращать взаимоблокировки, базируются на концепции безопасных состояний. Говорят, что состояние безопасно, если оно не находится в тупике и существует некоторый порядок планирования, при котором каждый процесс может работать до завершения, даже если все процессы вдруг захотят немедленно получить максимальное количество ресурсов.

Управление системой ввода-вывода

Аппаратура ввода-вывода

В настоящее время наблюдается все более и более активное развитие устройств ввода-вывода в компьютерных системах. В значительной степени это объясняется, во-первых, необходимостью ввода, обработки и вывода мультимедийной информации (аудио, видео, цифровых фотографий, отсканированных образов и других изображений), во-вторых, постоянной потребностью в увеличении скорости и емкости устройств вследствие гигантского роста размеров обрабатываемой информации. Еще в 1980-х гг., например, нормой считалось использование гибких дискет (*FDD*) емкостью 1.44 мегабайта для резервного копирования. Сейчас устройствами FDD настольные и портативные компьютеры вообще не комплектуются, а, что касается резервного копирования, то и устройств емкостью 128 гигабайт может оказаться недостаточно для этой цели.

Набор устройств включает, в частности:

- клавиатуру и мышь;
- жесткие диски (*HDD*), включая внутренние и внешние (*ZIV drives*);
- *flash*-память;
- ленточные стримеры;
- компакт-диски *BluRay*, *DVD*, *CD*;
- твердотельные накопители (*solid state drive – SSD*);
- устройства для мультимедийного ввода-вывода: порты и адаптеры *IEEE* 1394 (*Fire-Wire*) для подключения цифровых видеоустройств; порты и адаптеры *High Definition Multimedia Interface* (*HDMI*) для подключения видеоаппаратуры стандарта *High Definition (HD)*; кард-ридеры для нескольких форматов (*SmartMedia* и др.) носителей, используемых в цифровых фотоаппаратах;
- мониторы, видеокарты (видеоадаптеры) и графические процессоры, в том числе – многоядерные;
- принтеры, сканеры.

Основные концепции

Каждое устройство подключается к компьютерной системе через **порт** – контроллер и разъем (либо беспроводное устройство) для передачи данных между устройством ввода-вывода и компьютером. Каждый порт имеет свое традиционное обозначение и свой номер в системе. Порт может существовать физически, как разъем для проводного соединения и связанный с ним контроллер порта (например, *USB* – универсальный порт для подключения

широкого спектра устройств; *LPT* – порт для подключения принтеров и сканеров), либо может быть организован операционной системой как ***виртуальный порт*** для унификации обработки внешних устройств. Виртуальные порты, обычно – коммуникационные порты (*COM*) с большими номерами – например, *COM*10, *COM*15, - организуются для обмена с устройствами беспроводной связи – например, мобильными телефонами и органайзерами. Беспроводная связь чаще всего организуется через *Bluetooth* – радиосвязь на расстоянии до 20 м, в новых стандартах – до 1 км.

Шина* (*bus*)** - это цепочка устройств прямого доступа в компьютерной системе, через которую передается информация от одних устройств к другим. Обычно в настольных и портативных компьютерах используется шина ***PCI (Personal Computer Interface), тактовая частота которой в современных компьютерах 1 – 1.5 *GHz*. Она фактически и определяет суммарную производительность компьютерной системы. К шине *PCI* подключены контроллеры внешних устройств и портов.

***Контроллер* (*host adapter*)** – специализированный микропроцессор для управления внешним устройством и портом. Контроллер внешнего устройства – это устройство управления командами ввода-вывода с данным внешним устройством. Устройства имеют адреса, используемые командами непосредственного ввода-вывода и командами ввода-вывода, отображаемого в память. Каждый контроллер устройства использует свой буфер памяти для хранения одного или нескольких блоков информации, расположенный либо в специализированной памяти устройства (контроллера), либо являющийся частью оперативной памяти компьютерной системы.

Опрос устройств

Операционная система с помощью прерываний по таймеру организует **опрос устройств** – периодический анализ состояния каждого внешнего устройства. В процессе работы в состоянии устройств могли произойти изменения, например, пользователь установил флэшку в *USB*-порт, включил или выключил принтер и т.д. При опросе устройств ОС определяет состояние каждого устройства, которое может быть следующим:

- ***command-ready*** – готово к выполнению команд;
- ***busy*** – занято;
- ***error*** – ошибка.

При выполнении ввода-вывода аппаратура организует **цикл *busy-wait*** ожидания ввода-вывода с устройством: если устройство занято, процесс ждет его освобождения.

Линия ***запросов на прерывания*** **(*interrupt request – IRQ*)** переключается устройством ввода-вывода, которое сигнализирует с помощью запроса на прерывание о начале или окончании ввода-вывода.

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами.

1. *С помощью программируемого ввода-вывода*. В этом случае, когда процессору встречается команда, связанная с вводом-выводом, он выполняет ее, посылая соответствующие команды контроллеру ввода-вывода. Это устройство выполняет требуемое действие, а затем устанавливает соответствующие биты в регистрах состояния ввода-вывода и не посылает никаких сигналов, в том числе сигналов прерываний. Процессор периодически проверяет состояние модуля ввода-вывода с целью проверки завершения операции ввода-вывода. Недостатки такого метода – большие потери процессорного времени, связанные с управлением вводом-выводом.

2. *Ввод-вывод, управляемый прерываниями.* Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять текущий процесс, если нет необходимости в ожидании выполнения операции ввода-вывода. В противном случае текущий процесс приостанавливается до получения сигнала прерывания о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд. Такой ввод-вывод намного эффективнее, чем программируемый ввод-вывод, так как при этом исключается ненужное ожидание с бесполезным простоем процессора. Однако и в этом случае ввод-вывод потребляет еще значительное количество процессорного времени, потому что каждое слово, которое передается из памяти в модуль ввода-вывода (контроллер) или обратно, должно пройти через процессор.

3. *Прямой доступ к памяти* (*direct memory access – DMA*). В этом случае специальный модуль прямого доступа к памяти управляет обменом данных между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных. В настоящее время в персональных и других компьютерах используется третий способ ввода-вывода, поскольку в структуре компьютера имеется *DMA*-контроллер или подобное ему устройство, обслуживающее, как правило, запросы по передаче данных от нескольких устройств ввода-вывода на конкурентной основе.

DMA-контроллер имеет доступ к системной шине независимо от центрального процессора, как показано на рис.1. Контроллер содержит несколько регистров, доступных центральному процессу для чтения и записи (регистр адреса памяти, счетчик байтов, управляющие регистры). Управляющие регистры задают порт ввода-вывода, который должен быть использован, направление переноса данных (чтение или запись в устройство ввода-вывода),

единицу переноса (побайтно, пословно), а также число байтов, которые следует перенести за одну операцию.

Перед выполнением операции обмена ЦП программирует *DMA*-контроллер, устанавливая его регистры (шаг 1 на рис. 1). Затем ЦП дает команду дисковому контролеру прочитать внести данные во внутренний буфер и проверить контрольную сумму. После этого процессор продолжает свою работу. Когда данные получены и проверены контроллером диска, DMA может начинать работу.

DMA-контроллер начинает перенос данных, посылая дисковому контроллеру по шине запрос чтения (шаг 2). Адрес памяти уже находится на адресной шине, так что контроллер знает, куда пересылать следующее слово из своего буфера. Запись в память является еще одним стандартным циклом шины (шаг 3). Когда запись закончена, контроллер диска посылает сигнал подтверждения контролеру *DMA* (шаг 4). Затем контроллер *DMA* увеличивает используемый адрес памяти и уменьшает значение счетчика байтов. После этого шаги 2, 3 и 4 повторяются, пока значение счетчика не станет равным нулю. По завершению цикла копирования контроллер *DMA* инициирует прерывание процессора, сообщая ему о завершении операции ввода-вывода.

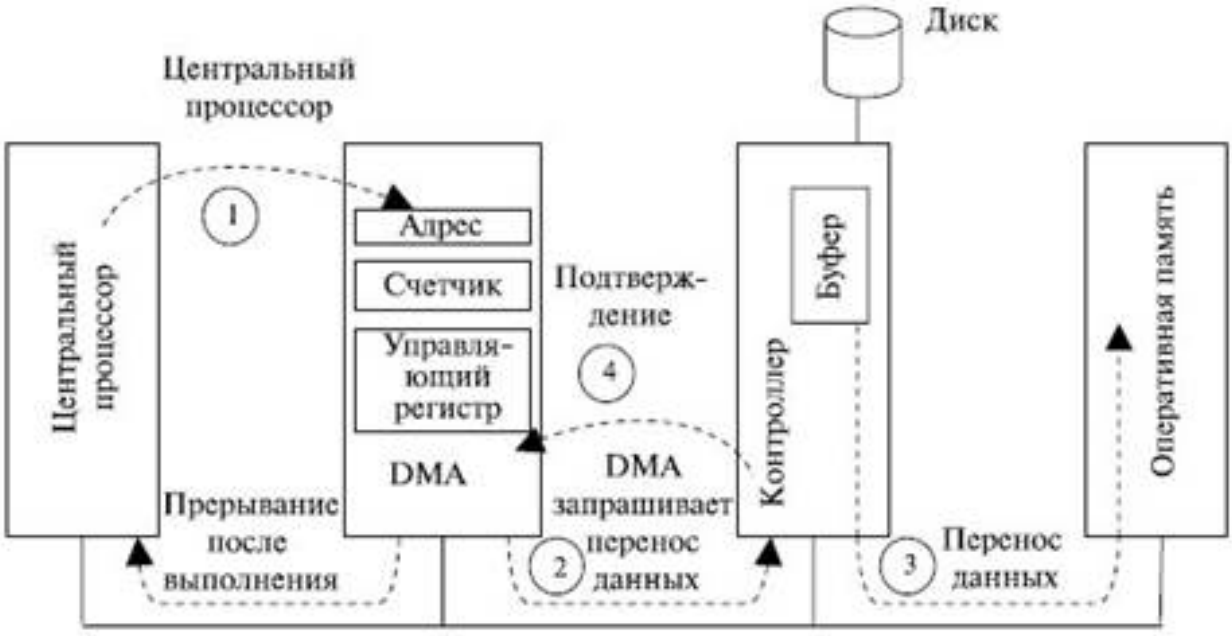


Рис. 1**.** Работа *DMA*

Блочные, символьные и сетевые устройства

Типичный пример ***блочного устройства*** – устройство управления дисками. Оно выполняет команды вида*:* ***read, write, seek*** (считать, записать или найти блок с заданным номером). Устройство может выполнять чистый ввод-вывод или

доступ к файловой системе. Имеется возможность доступа к файлу, отображаемому в память.

Типичные примеры ***символьных устройств*** – клавиатура, мышь, последовательные порты. Такие устройства выполняют команды вида: ***get, put*** (считать или записать символ).

Сетевые устройства существенно отличаются от блочных и символьных; имеют свой собственный интерфейс и систему команд. Сетевое устройство отделяет сетевой протокол от сетевой операции. Команды сетевых устройств включают функцию ***select*** – выбор сетевого пакета. Сетевые устройства различны по подходам к реализации (конвейеры, pipes, *FIFO*, потоки, очереди, почтовые ящики).

Блокируемый (синхронный) и не блокируемый (асинхронный) ввод-вывод

Для оптимизации ввода-вывода в системе поддерживается, помимо традиционного ***синхронного (блокируемого),*** также ***асинхронный*** ввод-вывод.

Блокируемый ввод-вывод основан на простой, интуитивно понятной парадигме: процесс задерживается, пока ввод-вывод не закончится. Он более прост для использования и понимания, но, в силу своей недостаточной эффективности, недостаточен для некоторых применений. Для оптимизации ввода-вывода возврат из системного вызова для ввода-вывода может происходит по мере доступности информации. Применяется пользовательский интерфейс для копирования данных (***буферизация***). Ввод-вывод также часть реализуется с помощью многопоточности (*multi-threading*): ввод-вывод выделяется в отдельный поток.

Асинхронный ввод-вывод основан на иной парадигме: процесс исполняется одновременно с выполнением ввода-вывода. Вследствие этого, он более сложен в использовании, так как большинство программистов до сих пор привыкли мыслить и реализовывать программы в последовательном стиле. После завершения асинхронного ввода-вывода подсистема ввода-вывода генерирует сигнал (исключение) в процессе, его использующем. Программирование асинхронного ввода-вывода основано на использовании ***пары*** операций типа ***начать асинхронный ввод-вывод*** *и* ***закончить асинхронный ввод-вывод*** (подождать его результатов). Асинхронный ввод-вывод обеспечивает наибольшую эффективность.

Супервизор ввода-вывода (компонента ядра ОС) решает следующие основные задачи:

- Осуществляет ***планирование***, включая упорядочение запросов на ввод-вывод в очередях к каждому устройству.

- Для балансировки устройств с разными скоростями и сглаживания несоответствия размера данных для работы с устройством обеспечивает ***буферизацию*** – запись данных в память в процессе передачи между устройствами.
- Как неоднократно отмечалось, для оптимизации работы с внешними устройствами организуется ***кэширование*** – использование быстрой память, в которой хранится копия данных (фактически в ней сохраняются наиболее часто используемые блоки).
- Весьма важна также такая функция ОС, как ***буферизация вывода*** **(*spooling*)** – задержка вывода на устройство, с целью поддержания целостности информации, выводимой одним и тем же процессом. Типичный пример – печать на принтер.
- Выполняет также ***резервирование устройства*** – обеспечение монопольного доступа к нему. Имеются системные вызовы для занятия и освобождения устройства монопольного доступа. ОС контролирует отсутствие тупиков (*deadlocks*), которые возможны при монопольном использовании устройств.
- Выполняет ***обработку ошибок*** ввода-вывода. Система поддерживает восстановление информации после чтения с диска, недоступности устройства, временных сбоев при записи.

Структуры данных для ввода-вывода в ядре ОС

В ядре ОС хранится информация о состоянии для компонент ввода-вывода, включая таблицы открытых файлов, сетевых соединений, состояние символьных устройств. Она представляет собой большое число сложных структур данных (очередей ввода-вывода и таблиц устройств) для контроля буферов, распределения памяти и др.

Каждая ОС имеет, по крайней мере, три системных таблицы:

1. Содержит информацию обо всех устройствах в/в, подключенных к вычислительной системе – ***таблица оборудования***, а каждый ее элемент – блок управления устройством в/в (***UCB*** – *unit control block*) содержит следующую информацию:
   - Тип устройства, конкретная модель, символическое имя, характеристика устройства.
   - Как это устройство подключено (тип интерфейса, через какой порт, линия запроса прерывания и т.д.).
   - Номер и адрес канала в/в.
   - Указание на драйвер, указание секции запуска и секции продолжения драйвера.

2. Предназначена для реализации еще одного принципа виртуализации устройств в/в – независимости от устройства. Это ***таблица описания виртуальных логических устройств*** (***DRT*** – *device reference table*). Она предназначена для установления связи между виртуальными (логическими) устройствами и реальными устройствами, описанными в таблице оборудования.

3. Необходима для организации обратной связи между ЦП и устройствами в/в – ***таблица прерываний***, указывает для каждого сигнала запроса на прерывание тот элемент *UCB*, который сопоставлен данному устройству, подключенному к этой линии запроса на прерывание.

С учетом изложенных принципов и таблиц, рассмотрим процесс управления в/в (рис. 2). От прикладной программы на супервизор программ поступает запрос на операцию в/в. Он проверяет системный вызов и в случае ошибки возвращает задаче соответствующее сообщение. Если запрос корректен, то он перенаправляется в супервизор ввода/вывода. Последний по логическому имени с помощью таблицы *DRT* находит соответствующий элемент *UCB* в таблице оборудования. Если устройство уже занято, то описатель задачи помещается в список задач, ожидающих настоящее устройство. Если же устройство свободно, то супервизор ввода/вывода определяет из *UCB* тип устройства и при необходимости запускает препроцессор, позволяющий получить последовательность управляющих кодов и данных, которую сможет понять и отработать устройство. Затем управление передается соответствующему драйверу на секцию запуска. Драйвер инициализирует операцию управления, обнуляет счетчик тайм-аута и возвращает управление диспетчеру задач с тем, чтобы он поставил на процессор готовую к исполнению задачу. Система работает своим чередом, но когда устройство в/в отработает посланную ему команду, оно выставляет сигнал запроса на прерывания, по которому через таблицу прерываний управление передается на секцию продолжения. Получив новую команду, устройство вновь начинает ее обрабатывать, а управление процессором опять передается диспетчеру задач, и процессор продолжает полезную работу. Таким образом, получается параллельная обработка задач, на фоне которой процессор осуществляет управление операциями ввода/вывода.

Если имеются специальные аппаратные средства для управления в/в, снимающие эту работу с центрального процессора (каналы прямого доступа к памяти), то в функции ЦП будут по-прежнему все рассмотренные выше шаги, за исключением последнего – непосредственного управления операциями в/в. В случае использования каналов прямого доступа к памяти последние используют канальные программы и разгружают ЦП, избавляя его от непосредственного обмена данными между памятью и внешними устройствами.

Задача, выдавшая запрос на операцию в/в, переводится супервизором в состояние ожидания завершения заказанной операции.

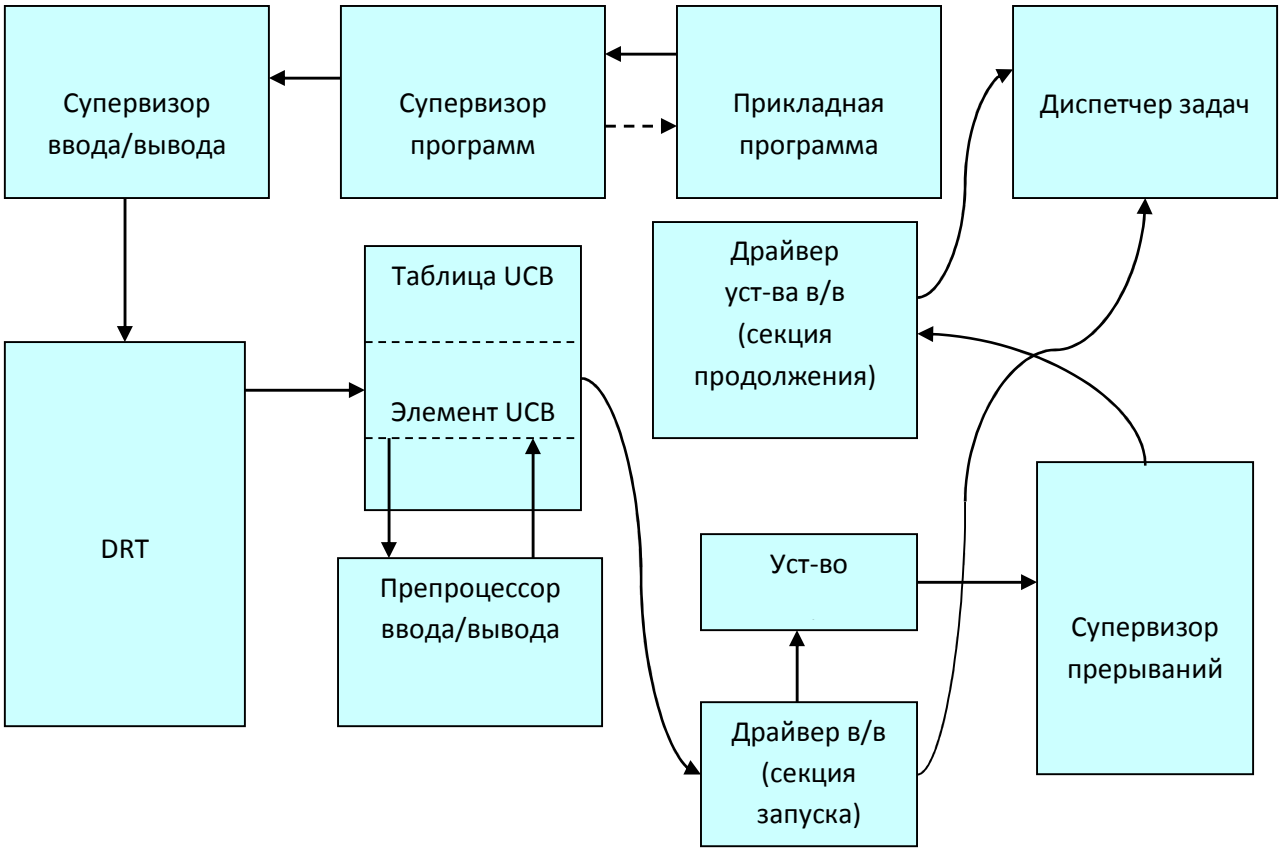


Рис. 2. Процесс управления вводом/выводом

Когда супервизор получает от секции завершения сообщение о том, что операция завершилась, он переводит задачу в состояние готовности к выполнению, и она продолжает свою работу. Эта ситуация соответствует синхронному в/в. Он является стандартным для большинства ОС.

Чтобы увеличить скорость выполнения приложений, в мультипрограммных ОС при необходимости используют асинхронный в/в.

Безопасность операционных систем и сетей

Введение

Безопасность – одна из наиболее актуальных проблем в области ИТ в настоящее время, ввиду сильной зависимости повседневной деятельности и бизнеса от компьютерных технологий и ввиду резко возрастающего числа сетевых атак (киберпреступности). Особенно важна безопасность для операционных систем и сетей как основных объектов атак.

Проблема безопасности

Безопасность (security) – это защита от внешних атак. В настоящее время наблюдается значительный рост числа самых разнообразных атак хакеров, угрожающих целостности информации, работоспособности компьютерных систем и зависящих от них компаний, благосостоянию и личной безопасности людей. Для защиты от атак необходимы специальные меры безопасности, компьютерные технологии и инструменты.

В любой компьютерной системе должна быть реализована ***подсистема безопасности***, которая должна проверять внешнее окружение системы и защищать ее от:

- Несанкционированного доступа.
- Злонамеренной модификации или разрушения.
- Случайного ввода неверной информации.

Практика показывает, что легче защитить от случайной, чем от злонамеренной порчи информации.

Аутентификация

Одной из наиболее широко используемых мер безопасности является ***аутентификация (authentication) –*** идентификация пользователей при входе в систему. Такая идентификация пользователей наиболее часто реализуется через ***логины –*** зарегистрированные имена пользователей для входа в систему – и ***пароли –*** секретные кодовые слова, ассоциируемые с каждым логином.

Основной принцип использования паролей в том, что они должны сохраняться в секрете. Поэтому одна из традиционных целей атакующих хакеров состоит в том, чтобы любыми способами выведать у пользователя его логин и пароль. Для сохранения секретности паролей предпринимаются следующие меры:

- ***Частая смена паролей***. Аналогичные меры применялись в армии во время войны. Большинство сайтов и других систем (например, сайт партнеров фирмы *Microsoft*) требуют от пользователей регулярной (например, не реже чем

раз в три месяца) смены паролей, иначе сайт блокируется для доступа. Подобные меры вполне оправданы.

- ***Использование "не угадываемых" паролей***. Практически все системы требуют от пользователя при регистрации устанавливать пароли, не являющиеся легко угадываемыми: например, как правило, пароль должен содержать большие и маленькие буквы и цифры, специальные символы и иметь длину не менее 7-8 символов. Используются также автоматические генераторы не угадываемых паролей. Поэтому использование в качестве паролей легко угадываемых слов – например, имени любимой собаки или общеупотребительного понятия – не рекомендуется.
- ***Сохранение всех неверных попыток доступа***. Во многих системах реализован системный журнал, в котором фиксируются все неверные попытки ввода логинов и паролей. Обычно дается фиксированное число таких попыток (например, три).

Программные угрозы (атаки)

Рассмотрим некоторые типичные виды угроз и атак, используемые хакерами.

Троянская программа (Trojan Horse) – атакующая программа, которая "подделывается" под некоторую полезную программу, но при своем запуске не по назначению (злонамеренно) использует свое окружение, например, получает и использует конфиденциальную информацию. Троянские программы используют системные механизмы для того, чтобы программы, написанные одними пользователями, могли исполняться другими пользователями.

Вход в ловушку (Trap Door) - использование логина или пароля, который позволяет избежать проверок, связанных с безопасностью.

Переполнение стека и буфера (Stack and Buffer Overflow) - использование ошибки в программе (переполнение стека или буферов в памяти) для обращения к памяти другого пользователя или процесса с целью нарушения ее целостности.

Системные угрозы (атаки)

Рассмотрим также некоторые типичные атаки, использующие ***уязвимости (vulnerabilities)*** в системных программах – ошибки и недочеты, дающие возможность организации атак.

Черви (Worms) – злонамеренные программы, использующие механизмы самовоспроизведения (размножения). Например, один из Интернет-червей использует сетевые возможности *UNIX* (удаленный доступ) и ошибки в программах ***finger*** *и* ***sendmail***. Принцип его действия следующий: некоторая постоянно используемая в сети системная программа распространяет главную программу червя.

Вирусы – фрагменты кода, встраивающиеся в обычные программы с целью нарушения работоспособности этих программ и всей компьютерной системы. В основном вирусы действуют на микрокомпьютерные системы. Вирусы скачиваются с публично доступных сайтов или с дисков, содержащих "инфекцию". Для предотвращения заражения компьютерными вирусами необходимо соблюдать принципы безопасности при использовании компьютеров (***safe computing***) – использовать ***антивирусы***, ***guards*** – программы, постоянно находящиеся в памяти и проверяющие на вирусы каждый открываемый файл - *.exe*, *doc*, и т.д.

Отказ в обслуживании (*Denial of Service – DoS*) – одна из распространенных разновидностей атак на сервер, заключающаяся в создании искусственной перегрузки сервера с целью препятствовать его нормальной работе. Например, для *Web*-сервера такая атака может заключаться в том, чтобы искусственно сгенерировать миллион запросов *"GET*". Если сервер реализован не вполне надежно, подобная атака чаще всего приводит к переполнению памяти на сервере и необходимости его перезапуска.

Типы сетевых атак

Рассмотрим некоторые типы современных сетевых атак, которых необходимо постоянно остерегаться пользователям.

Phishing **–** попытка украсть конфиденциальную информацию пользователя путем ее обманного получения от самого пользователя. Даже само слово ***phishing*** – искаженное слово ***fishing*** (рыбная ловля), т.е. хакер с помощью этого приема как бы пытается поймать чересчур наивного пользователя "на удочку". Например, напугав в своем сообщении пользователя, что его логин и пароль, кредитная карта или банковский счет под угрозой, хакер пытается добиться от пользователя в ответ ввода и отправки некоторой конфиденциальной информации. Обычно *phishing*-сообщение по электронной почте приходит как бы от имени банка и подделывается под цвета, логотипы и т.д., используемые на сайте банка. Однако для его разоблачения *обычно достаточно подвести курсор мыши (не кликая ее) к приведенной web*-ссылке или *email*-адресу (при этом она высвечивается) и убедиться в том, что адрес указывает отнюдь не на банк, а на совершенно посторонний сайт или *email*. Поэтому пользователям не следует быть слишком наивными. Другая действенная мера, если *phishing* происходит регулярно с одних и тех же *email*-адресов, – включить эти адреса в черный список на *email*-сервере. Тогда подобные сообщения вообще не будут доходить до входного почтового ящика пользователя.

Pharming **–** перенаправление пользователя на злонамеренный *Web*-сайт (обычно с целью *phishing*). Меры предотвращения со стороны пользователя мы уже рассмотрели. В современные *web*-браузеры встроены программы антифишингового контроля, которые запускаются автоматически при

обращении к сайту. Хотя это отнимает у пользователя некоторое время, подобные меры помогают предотвратить многие атаки.

Tampering with data – злонамеренное искажение или порча данных. Действенной мерой по борьбе с подобными атаками является ***криптование*** информации.

Spoofing – "подделка" под определенного пользователя (злонамеренное применение его логина, пароля и полномочий). Логин и пароль при этом либо получены от пользователя обманным путем (например, в результате *phishing*), либо извлечены из "взломанного" хакерской программой системного файла.

Elevation of privilege – попытка расширить полномочия (например, до полномочий системного администратора) с целью злонамеренных действий. Поэтому наиболее секретная информация в любой компьютерной системе – пароль системного администратора, который необходимо защищать особенно тщательно.

Принципы разработки безопасных программных продуктов

Новая схема жизненного цикла для разработки безопасных программ, разработанная и применяемая компанией Microsoft, носит название ***Security Development Life Cycle (SDLC).*** Основная идея *SDLC* – учитывать требования безопасности в течение всего жизненного цикла разработки программ, начиная с самых ранних этапов. Схема *SDLC* приведена на рис. 1.

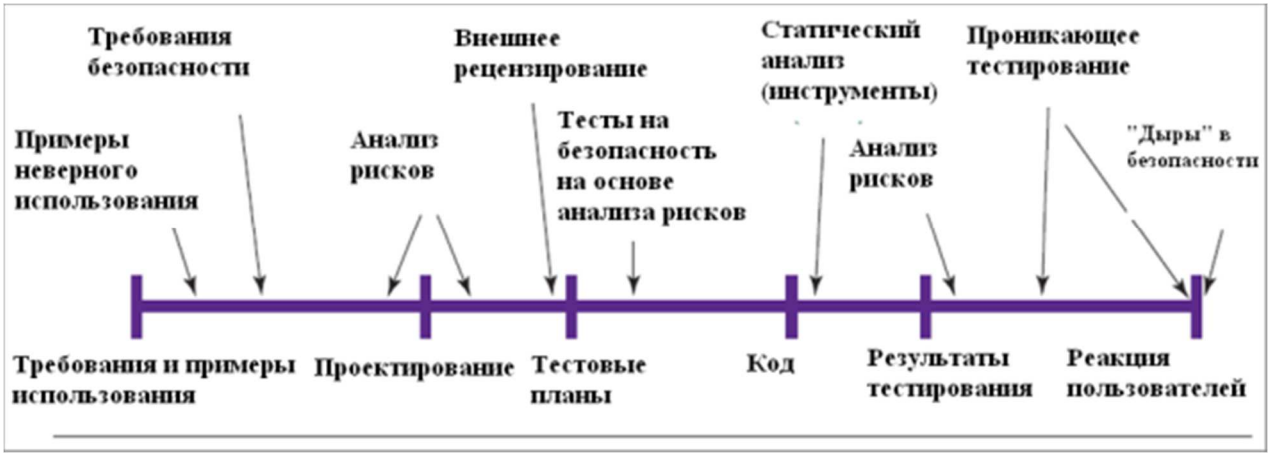


Рис. 1. Схема *Security Development Life Cycle (SDLC)*

На схеме приведены традиционные этапы жизненного цикла и стрелками показаны дополнения к ней, которые вносятся *SDLC*. В течение всего цикла разработки ПО, начиная с самых ранних этапов (требования, спецификации, проектирование), необходимо постоянно предусматривать меры надежности и безопасности ПО, чтобы впоследствии не пришлось их встраивать в систему в "авральном порядке", что значительно увеличит затраты.

Компания *Microsoft* предложила ряд простых схем для оценки и разработки безопасного программного обеспечения, для оценки угроз и оценки последствий атаки.

Брандмауэр

Брандмауэр (firewall) **–** системное программное обеспечение для защиты локальной сети от внешних атак. Брандмауэр помещается между "доверенными" и "не доверенными" компьютерами – например, компьютерами данной локальной сети и всеми остальными. Брандмауэр ограничивает сетевой доступ между двумя различными доменами безопасности. Встроенный брандмауэр имеется во всех современных операционных системах и по умолчанию включен. Настоятельно рекомендуется не отключать его, что особенно существенно при выходе в Интернет.

Схема использования брандмауэра изображена на рис. 2.

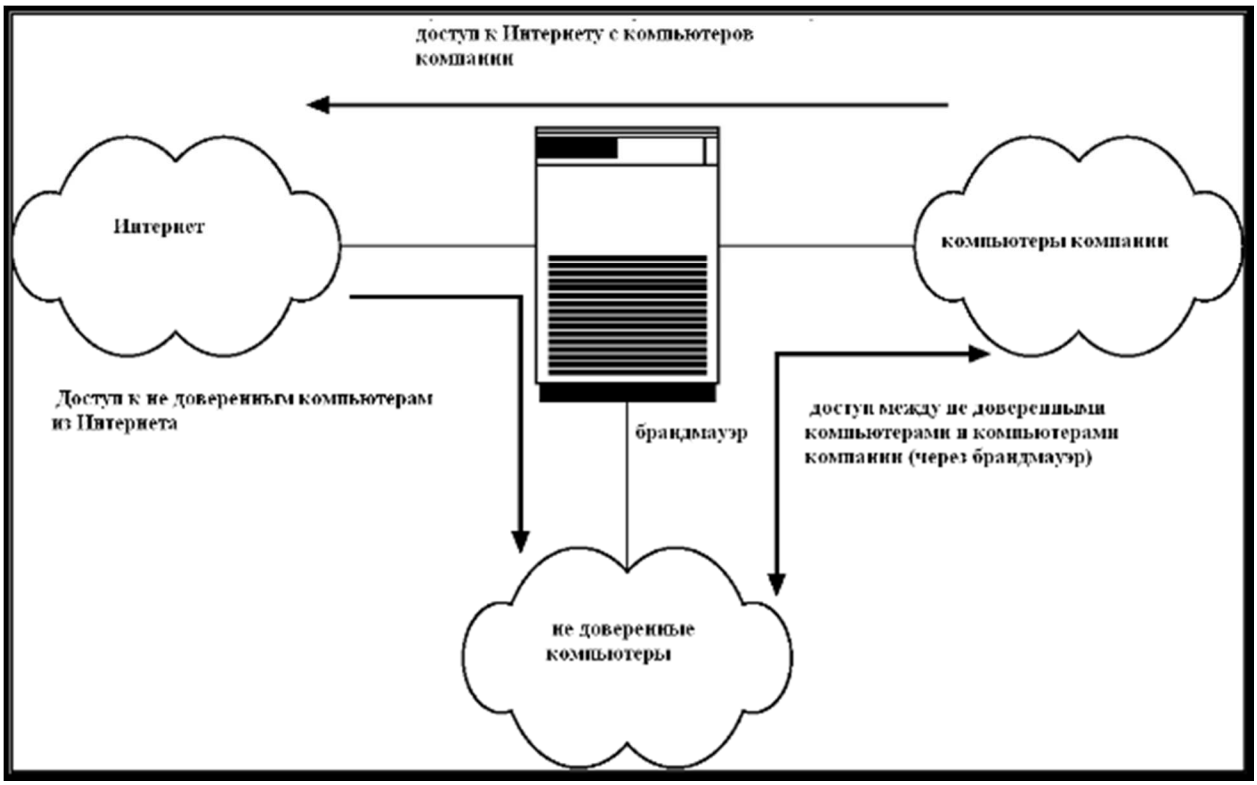


Рис. 2. Схема использования брандмауэра для обеспечения сетевой безопасности.

Реализация брандмауэров, как правило, основана на фильтрации сетевых пакетов, пересылаемых с "надежных" и потенциально ненадежных *IP*-адресов.

Обнаружение попыток взлома

Действенной мерой безопасности является обнаружение попыток входа в компьютерные системы. Методы обнаружения:

- Аудит и ведение журнала.
- Использование ***tripwire*** - программ для *UNIX*, которые проверяют, не изменялись ли некоторые файлы и директории, например, файлы, содержащие пароли;
- Слежение за системными вызовами.

Криптография

Криптография – это преобразование понятного текста в зашифрованный текст. Методы криптографии широко используются для защиты конфиденциальной информации.

Свойства хороших методов криптования:

- Относительно простой для авторизованных пользователей способ криптования и декриптования данных.
- Схема криптования должна зависеть не от секретного алгоритма, а от секретного параметра алгоритма, называемого ***ключом криптования (encryption key***).
- Для несанкционированного пользователя должно быть крайне сложно определить ключ.

Технология ***Data Encryption Standard (DES)*** основана на подстановке символов и изменении их порядка на основе ключа, предоставляемого авторизованным пользователям через защищенный механизм. Такая схема лишь настолько безопасна, насколько безопасен сам механизм получения ключа.

Криптование на основе открытого ключа – другой широко распространенный метод криптографии - основано на принципе, при котором пользователю известны два ключа:

public key – открытый ключ для криптования данных.

private key – закрытый ключ, известный только пользователю и применяемый им для декриптования данных.

Метод открытого ключа воплощает еще одно важное требование к методам криптографии: метод должен быть основан на схеме криптования, которая публично доступна, но это не будет облегчать разгадывание схемы декриптования.

Примером криптования, используемым в *Web*-технологиях, является ***SSL (Secure Socket Layer)*** – семейство криптографических протоколов, предназначенное для обмена криптованными сообщениями через сокет. *SSL* используется для защищенного взаимодействия между *Web*-серверами и браузерами (например, ввода номеров кредитных карт). При обращении клиента к серверу последний проверяется с помощью сертификата. Взаимодействие

между компьютерами использует криптографию на основе симметричного ключа.

Уровни безопасности компьютеров

Министерство обороны США классифицирует безопасность компьютеров по уровням: *A, B, C, D*.

Уровень безопасности D соответствует минимальному уровню безопасности.

На ***уровне безопасности C*** обеспечиваются периодические проверки компьютера с помощью аудита. Уровень *C* подразделяется на уровни *C*1 и *C*2. **Уровень *C*1** обозначает взаимодействие пользователей с одинаковым уровнем безопасности. **Уровень *C*2** допускает управление доступом на уровне пользователей.

Уровень безопасности B имеет все свойства уровня *C*, однако каждый объект может иметь уникальные ***метки чувствительности (sensitivity labels).*** Подразделяется на уровни *B*1, *B*2, *B3*.

На ***уровне безопасности A*** используются формальные методы спецификации и проектирования для обеспечения безопасности.

Анализ некоторых популярных ОС с точки зрения их защищенности

Итак, ОС должна способствовать реализации мер безопасности или непосредственно поддерживать их. Примерами подобных решений в рамках аппаратуры и операционной системы могут быть:

- разделение команд по уровням привилегированности;
- сегментация адресного пространства процессов и организация защиты сегментов;
- защита различных процессов от взаимного влияния за счет выделения каждому своего виртуального пространства;
- особая защита ядра ОС;
- контроль повторного использования объекта;
- наличие средств управления доступом;
- структурированность системы, явное выделение надежной вычислительной базы (совокупности защищенных компонентов), обеспечение компактности этой базы;
- следование принципу минимизации привилегий – каждому компоненту дается ровно столько привилегий, сколько необходимо для выполнения им своих функций.

Большое значение имеет структура файловой системы. Например, в ОС с дискреционным (избирательным) контролем доступа каждый файл должен храниться вместе с дискреционным списком прав доступа к нему, а, например,

при копировании файла все атрибуты должны быть автоматически скопированы вместе с телом файла.

В принципе, меры безопасности не обязательно должны быть заранее встроены в ОС – достаточно принципиальной возможности дополнительной установки защитных продуктов. Тем не менее по-настоящему надежная система должна изначально проектироваться с акцентом на механизмы безопасности.

MS-DOS

ОС MS-DOS функционирует в реальном режиме (*real-mode*) процессора i80x86. В ней невозможно выполнение требования, касающегося изоляции программных модулей (отсутствует аппаратная защита памяти). Уязвимым местом для защиты является также файловая система *FAT*, не предполагающая у файлов наличия атрибутов, связанных с разграничением доступа к ним. Таким образом, *MS-DOS* находится на самом нижнем уровне в иерархии защищенных ОС.

OS/2

OS/2 работает в защищенном режиме (*protected-mode*) процессора i80x86. Изоляция программных модулей реализуется при помощи встроенных в этот процессор механизмов защиты памяти. Поэтому она свободна от указанного выше коренного недостатка систем типа *MS-DOS*. Но *OS*/2 была спроектирована и разработана без учета требований по защите от несанкционированного доступа. Это сказывается прежде всего на файловой системе. Кроме того, пользовательские программы имеют возможность запрета прерываний. Следовательно, сертификация *OS*/2 на соответствие какому-то классу защиты не представляется возможной.

Unix

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. В *Unix* есть несколько уязвимых с точки зрения безопасности мест, хорошо известных опытным пользователям, вытекающих из самой природы *Unix*. Однако хорошее системное администрирование может ограничить эту уязвимость.

Относительно защищенности *Unix* сведения противоречивы. В *Unix* изначально были заложены идентификация пользователей и разграничение доступа. Как оказалось, средства защиты данных в *Unix* могут быть доработаны, и сегодня можно утверждать, что многие клоны *Unix* по всем параметрам соответствуют классу безопасности *C*2.

В *Unix* существует список именованных пользователей, в соответствии с которым может быть построена система разграничения доступа.

В ОС *Unix* считается, что информация, нуждающаяся в защите, находится главным образом в файлах.

По отношению к конкретному файлу все пользователи делятся на три категории:

- владелец файла;
- члены группы владельца;
- прочие пользователи.

Для каждой из этих категорий режим доступа определяет права на операции с файлом, а именно:

- право на чтение;
- право на запись;
- право на выполнение (для каталогов – право на поиск).

В итоге девяти (3х3) битов защиты оказывается достаточно, чтобы специфицировать *ACL* (*Access Control List* – список управления доступом) каждого файла.

Аналогичным образом защищены и другие объекты ОС *Unix*, например, семафоры, сегменты разделяемой памяти и т. п.

Среди всех пользователей особое положение занимает пользователь *root*, обладающий максимальными привилегиями. Обычные правила разграничения доступа к нему не применяются – ему доступна вся информация на компьютере.

В *Unix* имеются инструменты системного аудита – хронологическая запись событий, имеющих отношение к безопасности. К таким событиям обычно относят: обращения программ к отдельным серверам; события, связанные с входом/выходом в систему и другие. Обычно регистрационные действия выполняются специализированным *syslog*-демоном, который проводит запись событий в регистрационный журнал в соответствии с текущей конфигурацией. *Syslog*-демон стартует в процессе загрузки системы.

Таким образом, безопасность ОС *Unix* может быть доведена до соответствия классу С2. Однако разработка на ее основе автоматизированных систем более высокого класса защищенности может быть сопряжена с большими трудозатратами.

Windows NT/2000/*XP*

С момента выхода версии 3.1 осенью 1993 года в *Windows NT* гарантировалось соответствие уровню безопасности *C*2. В настоящее время (точнее, в 1999 г.) сертифицирована версия *NT* 4 с *Service Pack* 6*a* с использованием файловой системы *NTFS* в автономной и сетевой конфигурации. Следует помнить, что этот уровень безопасности не подразумевает защиту информации, передаваемой по сети, и не гарантирует защищенности от физического доступа.

Компоненты защиты *NT* частично встроены в ядро, а частично реализуются подсистемой защиты. Подсистема защиты контролирует доступ и учетную информацию. Кроме того, *Windows NT* имеет встроенные средства, такие как поддержка резервных копий данных и управление источниками бесперебойного питания, которые не требуются "Оранжевой книгой", но в целом повышают общий уровень безопасности.

ОС *Windows* 2000 сертифицирована по стандарту *Common Criteria*. В дальнейшем линейку продуктов *Windows NT*/2000/*XP*, изготовленных по технологии *NT*, будем называть просто *Windows NT*.

Ключевая цель системы защиты *Windows NT* – следить за тем, кто и к каким объектам осуществляет доступ. Система защиты хранит информацию, относящуюся к безопасности для каждого пользователя, группы пользователей и объекта. Единообразие контроля доступа к различным объектам (процессам, файлам, семафорам и др.) обеспечивается тем, что с каждым процессом связан маркер доступа, а с каждым объектом – дескриптор защиты. Маркер доступа в качестве параметра имеет идентификатор пользователя, а дескриптор защиты – списки прав доступа. ОС может контролировать попытки доступа, которые производятся процессами прямо или косвенно инициированными пользователем.

Windows NT отслеживает и контролирует доступ как к объектам, которые пользователь может видеть посредством интерфейса (такие, как файлы и принтеры), так и к объектам, которые пользователь не может видеть (например, процессы и именованные каналы). Любопытно, что, помимо разрешающих записей, списки прав доступа содержат и запрещающие записи, чтобы пользователь, которому доступ к какому-либо объекту запрещен, не смог получить его как член какой-либо группы, которой этот доступ предоставлен.

Система защиты ОС *Windows NT* состоит из следующих компонентов:

- Процедуры регистрации (*Logon Processes*), которые обрабатывают запросы пользователей на вход в систему. Они включают в себя начальную интерактивную процедуру, отображающую начальный диалог с пользователем на экране и удаленные процедуры входа, которые позволяют удаленным пользователям получить доступ с рабочей станции сети к серверным процессам *Windows NT*.
- Подсистемы локальной авторизации (*Local Security Authority, LSA*), которая гарантирует, что пользователь имеет разрешение на доступ в систему. Этот компонент – центральный для системы защиты *Windows NT*. Он порождает маркеры доступа, управляет локальной политикой безопасности и предоставляет интерактивным пользователям аутентификационные услуги. *LSA* также контролирует политику аудита и ведет журнал, в котором сохраняются сообщения, порождаемые диспетчером доступа.
- Менеджера учета (*Security Account Manager, SAM),* который управляет базой данных учета пользователей. Эта база данных содержит информацию обо

всех пользователях и группах пользователей. *SAM* предоставляет услуги по легализации пользователей, применяющиеся в *LSA*.

- Диспетчера доступа (*Security Reference Monitor, SRM*), который проверяет, имеет ли пользователь право на доступ к объекту и на выполнение тех действий, которые он пытается совершить. Этот компонент обеспечивает легализацию доступа и политику аудита, определяемые *LSA*. Он предоставляет услуги для программ супервизорного и пользовательского режимов, для того чтобы гарантировать, что пользователи и процессы, осуществляющие попытки доступа к объекту, имеют необходимые права. Данный компонент также порождает сообщения службы аудита, когда это необходимо.

Microsoft Windows NT – относительно новая ОС, которая была спроектирована для поддержки разнообразных защитных механизмов, от минимальных до *C*2, и безопасность которой наиболее продумана. Дефолтный уровень называется минимальным, но он легко может быть доведен системным администратором до желаемого уровня.

Заключение

Решение вопросов безопасности операционных систем обусловлено их архитектурными особенностями и связано с правильной организацией идентификации и аутентификации, авторизации и аудита.

Наиболее простой подход к аутентификации – применение пользовательского пароля. Пароли уязвимы, значительная часть попыток несанкционированного доступа в систему связана с компрометацией паролей.

Авторизация связана со специфицированием совокупности аппаратных и программных объектов, нуждающихся в защите. Для защиты объекта устанавливаются права доступа к нему. Набор прав доступа определяет домен безопасности. Формальное описание модели защиты осуществляется с помощью матрицы доступа, которая может храниться в виде списков прав доступа или перечней возможностей.

Аудит системы заключается в регистрации специальных данных о различных событиях, происходящих в системе и так или иначе влияющих на состояние безопасности компьютерной системы.

Среди современных ОС вопросы безопасности лучше всего продуманы в ОС *Windows NT*.