Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Оренбургский государственный университет»

А. Ю. Кручинин, Р. Р. Галимов, А. А. Рычкова

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

Рекомендовано ученым советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательной программе высшего образования по направлению подготовки 10.03.01 Информационная безопасность

Оренбург
2019

УДК 004(07)
ББК 32.81я7
К 84

Рецензент - доцент, кандидат технических наук Ю. И. Синицын

Кручинин А.Ю.
К 84 Операционные системы [Электронный ресурс] : учебное пособие / А.Ю. Кручинин, Р.Р. Галимов., А.А. Рычкова – Оренбург: ОГУ, 2019. – 152 с.
ISBN 978-5-7410-2306-8

В учебном пособии рассмотрены основные принципы современных операционных систем – управление процессами и потоками, управление памятью, управление вводом-выводом и файловые системы, основы безопасности операционных систем, архитектуры существующих операционных систем.

Учебное пособие предназначено для студентов по направлению подготовки 10.03.01 Информационная безопасность по дисциплинам «Операционные системы и администрирование средств защиты информации» и «Многопользовательские операционные системы».

УДК 004(07)
ББК 32.81я7

ISBN 978-5-7410-2306-8

Содержание

Введение

Важным элементом компьютера является операционная система (ОС), позволяющая значительно упростить взаимодействие пользователя с аппаратными ресурсами. Операционная система компьютера представляет собой комплекс взаимосвязанных программ, который действует как интерфейс между приложениями и пользователями, с одной стороны, и аппаратурой компьютера, с другой стороны [16]. ОС выполняет две основные функции:

- обеспечивает удобный унифицированный интерфейс пользователям и программистам для взаимодействия с ресурсами ПК посредством предоставления виртуальной машины вместо реального аппаратного обеспечения;
- эффективное распределение ресурсов компьютера между множеством программ.

Фактически пользователя интересует только возможности эффективного использования прикладных программ компьютера, а не устройство персонального компьютера. Операционная система позволяет упросить пользователю запуск прикладных приложений, так как выполняет множество необходимых действий: загрузку кода и данных в оперативную память, инициализацию устройств ввода/вывода, подготовку ресурсов. Также ОС значительно облегчает процесс разработки программ. Это определяется тем, что множество функций для работы с аппаратным обеспечением (обработка прерываний, взаимодействие с видеокартой, жестким диском) реализовано операционной системой. При написании кода можно использовать данные функции с помощью специальных команд, называемых системными вызовами. Поэтому современные приложения включает в себя множества системных вызовов, необходимых, например, для работы с файлами. Таким образом, ОС скрывает от программиста конкретную реализацию аппаратного устройства компьютера и предоставляет общий интерфейс для взаимодействия. В частности, программист при выводе графической информации при помощи средств ОС на устройство ввода-вывода не знает конкретной модели видеокарты или принтера, их системы команд. Но при этом использует общие команды

графического интерфейса пользователя GUI, которые обрабатывает ОС и преобразует при помощи драйверов на «язык» конкретного устройства. Это позволяет разрабатывать приложения без привязки только к определенным моделям аппаратных устройств.

Современные компьютеры реализуют многозадачный режим работы, обуславливающий задачу распределения общих ресурсов, таких как время процессоров, памяти, устройств ввода-вывода между множеством одновременно исполняемых программ. Таким образом, операционная система является также менеджером ресурсов.

Функционирование операционной системы имеет следующие особенности:

- функции ОС так же, как и остального ПО, реализуются в виде отдельных программ или набора программ, исполняющихся процессов;
- ОС должна передавать управление другим процессам и ожидать, когда процессор снова выделит ей время для выполнения своих обязанностей.

Управление ресурсами включает решение следующих общих, не зависящих от типа ресурса задач:

- планирование ресурса – определение, какому процессу, когда и в каком количестве (если ресурс может выделяться частями) следует выделить данный ресурс;
- удовлетворение запросов на ресурсы;
- отслеживание состояния и учет использования ресурса – поддержание оперативной информации о том, занят или свободен ресурс и какая его доля уже распределена;
- разрешение конфликтов между процессами [16].

Управление ресурсами в современных операционных системах осуществляется двумя основными способами:

- мультиплексирование во времени. В данном случае доступ к ресурсу осуществляется по очереди в разные моменты времени. Примером является распределение времени процессора между потоками;

-мультиплексирование в пространстве. В данном способе каждый потребитель

получает определенную часть ресурса. При этом данные участки не пересекаются. Подобным образом происходит распределение памяти между программами [20].

История развития операционных систем непосредственно связана с историей развития компьютеров, которая начинается в середине девятнадцатого века с разработкой английским математиком Чарльзом Бэббиджем механической «аналитической машины», которая правда так и не заработала должным образом. Далее приведены данные о поколениях компьютеров и их связь с операционными системами с учетом их производительности и сложности разработки для них программ.

Первое поколение 1945-1955. Компьютеры состояли в основном из электронных ламп, и только некоторые из них можно было программировать. Программы разрабатывались на машинном коде и реализовывались в виде коммутационных панелей или перфокарт. Программист устанавливал коммутационную панель в вычислительное устройство и ожидал результатов вычисления несколько часов. Операционная система в данных компьютерах не использовалась.

Второе поколение 1955-1965.

Изобретение транзисторов позволило создавать относительно надежные и сложные вычислительные устройства, которые назывались мэйнфреймами. Данные компьютеры занимали целые комнаты. Для уменьшения затрат на загрузку программ применяется механизм пакетной обработки, который предполагает формирование из множества перфокарт общего задания и перенос на магнитную ленту, используя недорогой компьютер. Сформированное пакетное задание на магнитной ленте передается на исполнение мэйнфрейму. Появились операционные системы Fortran Monitor System (FMS), IBSYS. В качестве языков программирования использовались Фортран и Ассемблер.

Третье поколение 1965-1980.

В данный период фирма IBM впервые в своих компьютерах применила интегральные схемы. Наряду с большими компьютерами широкое распространение получили небольшие устройства, использующиеся коммерческими организациями.

IBM создает различные серии компьютеров, начиная с IBM/360, предназначенных как для выполнения небольших расчетов в коммерческих организациях, так и для проведения сложных научных и военных вычислений. Для них была разработана громоздкая ОС OS/360, которая в 1000 раз превышала по величине FMS второго поколения. Характерной особенностью данного периода времени является реализация многозадачности, заключающейся в возможности процессора переключаться на выполнение другой программы когда текущая приостанавливается в ожидании завершения работы устройства ввода-вывода. Другими известными операционными системами этого периода являются CTSS (совместимая система разделения времени) и MULTICS (мультиплексная информационная и вычислительная служба), которая была предназначена для обеспечения доступа сразу для сотни пользователей к одной машине. Дальнейшее развитие данной системы переросло в UNIX.

Четвёртое поколение: 1980-наши дни.

Производители компьютеров начали использовать большие интегральные схемы. В 1974 году Intel разработал первый универсальный 8-разрядный процессор i8080, который является предшественником intel 8086. Данный период характеризуется тем, что широкое распространение получили недорогие компьютеры, которые могли позволить себе обычные люди. Так в начале 80-х IBM выпустила персональный компьютер IBM PC. В этот же период времени была разработана операционная система MS-DOS, развитие которой привело к появлению операционных систем семейства Windows. Большинство ОС в это время поддерживали консольный интерфейс, когда взаимодействие с пользователем осуществлялось в текстовом режиме. Это требовало, чтобы пользователи обладали высоким уровнем знаний системы команд ОС. Одним из первых реализовала дружественный графический интерфейс фирма Macintosh в компьютерах Apple. Это обеспечило возможность работать с компьютерами не только специалистам, но и обычным людям. Под влиянием её успехов корпорация Microsoft выпускает графическую оболочку для MS-DOS – Windows. В 1995 году была разработана уже новая ОС Windows 95, которая положила начало развитию операционных систем

данного семейства: Windows 98, Windows NT, Windows 2000, XP, Windows 7, Windows 10 и другие. В классе сетевых и серверных компьютеров в это время в основном используется ОС UNIX. Альтернативой Windows в классе персональных компьютеров становится Linux.

Пятое поколение: 1990-наши дни.

С середины 1990-х начинается выпуск смартфонов, объединяющих в себе сотовый телефон и КПК. Для них были выпущены различные операционные системы: Symbian OS, RIM Blackberry OS, Windows Phone. На сегодняшний день наиболее популярными являются ОС Android и Apple iOS.

Таким образом, на сегодняшний день существует большое количество ОС. Далее представлена классификация операционных систем по области применения.

Операционные системы мэйнфреймов.

Мэйнфреймы характеризуются большими вычислительными ресурсами: большим количеством процессоров, каналов ввода-вывода, значительным объемом оперативной и внешней памяти. Компьютеры данного класса в основном используются для обработки больших объемов данных, запросов. Раньше подобные компьютеры занимали целые комнаты. Современные мэйнфреймы, например IBM — System z13, по размеру соответствуют шкафу. ОС для компьютеров данного класса предназначены на одновременную обработку большого количества задач, которые осуществляют огромное количество операций ввода-вывода. Система должна обеспечить обработку тысячи запросов в секунду. Примером является операционная система IBM z/OS.

Серверные операционные системы.

Серверы обладают более низкой производительностью в сравнении с мэйнфреймами и представляют собой мощный ПК или рабочую станцию. Данные компьютеры могут выполнять разнообразные функции: общий доступ к аппаратным и программным ресурсам (файлы, принтеры и другие), организация веб-серверов, прокси-серверов. Примерами подобных ОС являются Unix, Linux, Windows Server различных версий, FreeBSD и др.

Многопроцессорные операционные системы.

Данные системы применяются на компьютерах с несколькими центральными процессорами. Для них требуются специальные операционные системы, но обычно они представляют собой модификации серверных операционных систем.

Операционные системы для персональных компьютеров.

Основным критерием эффективности ОС данного класса является обеспечение удобного интерфейса для пользователя, позволяющего быстро научиться работать с компьютером. Популярными операционными системами для персональных компьютеров являются Windows 98, 2000, XP, 7, 10, Macintosh, Linux.

Операционные системы реального времени.

Системы реального времени в основном используются в промышленности для контроля и управления технологическими процессами, где вычислительные устройства должны обрабатывать запросы за ограниченный интервал времени. Несоблюдение данного требования может привести к техногенным катастрофам. Различают два основных вида подобных ОС: системы жесткого или мягкого режима работы. Первые гарантируют выполнение запроса за заданный интервал времени, вторые – просто обеспечивают высокую скорость отклика. К ОС реального времени (Real Time Operating Systems) относятся VxWorks, QNX.

Встроенные операционные системы.

К ним относятся ОС «карманных компьютеров» PDA (Personal Digital Assistant – персональный цифровой помощник). Кроме того, встроенные системы работают на машинах, в телевизорах, мобильных телефонах. В этих операционных системах обычно присутствуют все характеристики ОС реального времени с ограничением памяти, производительности и т.п. Примеры встроенных систем – PalmOS, Windows CE.

Операционные системы для смарт-карт.

Смарт-карта – устройство размером с кредитную карту, содержащее процессор. На такие системы налагаются жесткие ограничения по мощности и объему памяти. Некоторые ОС управляют только одной операцией, например, обеспечение электронных платежей. Отдельные смарт-карты включают поддержку виртуальной машины Java.

1 Архитектура операционной системы

1.1 Классическая архитектура

Множество модулей операционной системы делятся на две основные группы:

- ядро – модули, выполняющие основные функции операционной системы;
- модули, выполняющие вспомогательные функции операционной системы.

Модули ядра реализуют наиболее важные функции ОС, например, планирование работы процессов, управление памятью, устройствами ввода-вывода и т.п. Ядро является основой операционной системы, без которой она неработоспособна.

Вспомогательные модули ОС делятся на следующие классы:

- утилиты – это программы, решающие отдельные задачи по улучшению и сопровождению компьютерной системы, такие как программы архивирования или восстановления данных, очистки системы от ненужных файлов, средства для создания и изменения логических дисков и множество других;
- системные обрабатывающие программы – текстовые или графические редакторы, компиляторы, компоновщики, отладчики;
- программы предоставления пользователю дополнительных услуг – специальный вариант пользовательского интерфейса, калькулятор и даже игры;
- библиотеки процедур различного назначения, упрощающие разработку приложений, например, библиотеки математических функций, функций ввода-вывода и т. д.

Для обеспечения надежной работы операционная система должна работать в привилегированном режиме по отношению к остальным программам. Это необходимо, чтобы обезопасить ОС от воздействия некорректно работающих или вредоносных программ, например, от попыток модифицировать системный код.

Реализация привилегий операционной системы невозможно без аппаратной

поддержки. Современные ОС для этого используют защищенный режим работы процессора. Аппаратные средства ПК должны обеспечивать работу в двух режимах — пользовательском (user mode) и привилегированном, который называют режимом ядра (kernel mode). На рисунке 1 представлено разделение программного обеспечения компьютера по режиму их работы.

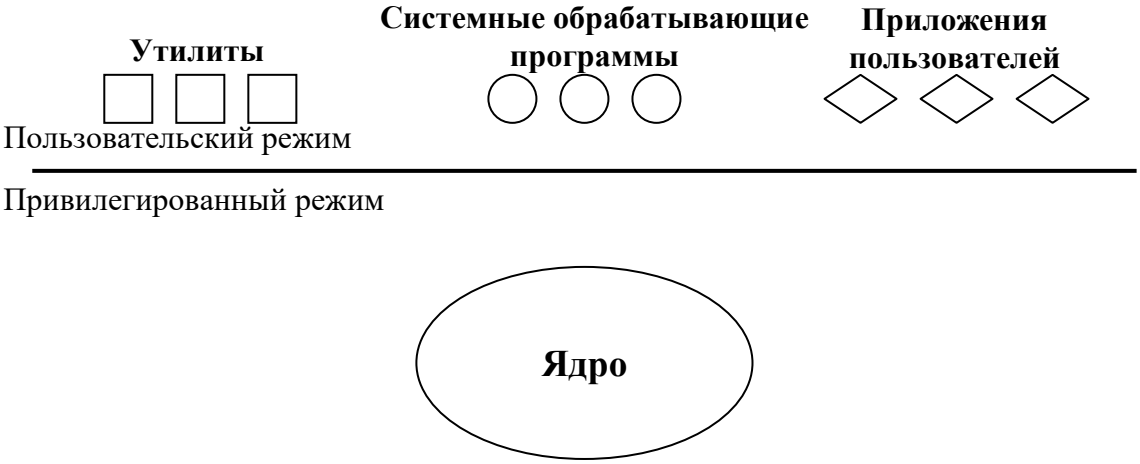


Рисунок 1 – Архитектура операционной системы с ядром в привилегированном режиме

Программы находятся в подчиненном положении относительно операционной системы за счет запрета исполнения в пользовательском режиме критичных команд, связанных с переключением процессора с задачи на задачу, управлением устройствами ввода-вывода, доступом к механизмам распределения и защиты памяти.

При этом уровней привилегий может быть несколько – 2, 3, 4 и т.д. Причем нет жесткой связи между количеством уровней привилегий, реализуемых аппаратно или средствами операционной системы. Например, процессоры фирмы Intel обеспечивают возможность реализации четырех уровней привилегий. При этом операционная система OS/2 организует трехуровневую систему привилегий для данных компьютеров, а Windows NT, UNIX и некоторые другие только двухуровневую.

Повышение степени защищенности ОС, обеспечиваемое реализацией многоуровневой системы привилегий, достигается за счет замедления исполнения системных вызовов. Системный вызов – это механизм, позволяющий прикладным

программам запросить у ядра ОС выполнение определенных услуг. Запрос прикладной программы на выполнение системного вызова инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению – происходит переключение обратно (Рисунок 2). Дополнительные служебные издержки выполнения системного вызова вследствие двукратной смены режима приводят к тому, что он выполняется медленнее, чем вызов процедуры без смены режима.

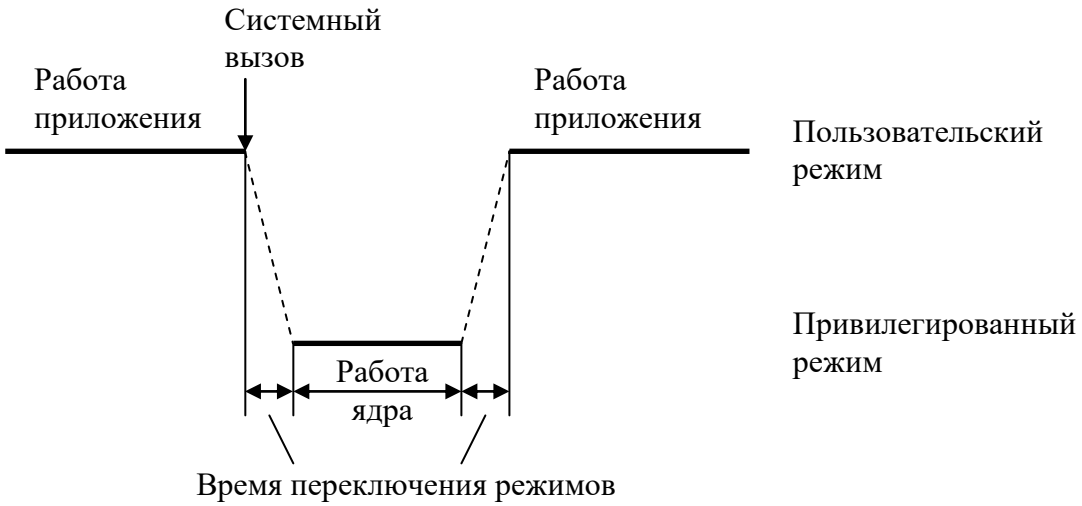


Рисунок 2 – Временная диаграмма исполнения системного вызова

Вычислительная машина, использующая операционную систему на основе ядра, представляет собой иерархическую трехуровневую систему: нижний уровень образуют аппаратные средства, промежуточный – ядро ОС, утилиты, обрабатывающие программы и приложения - верхний слой. Каждый уровень компьютера обслуживает вышестоящий, выполняя для него некоторый набор функций, образующий межслойный интерфейс.

Ядро ОС - это сложный многофункциональный аппаратно-программный комплекс. Также его можно рассматривать как многоуровневую систему, состоящую из следующих элементов:

- *Средства аппаратной поддержки операционной системы.* К ОС относятся только те средства, которые непосредственно обеспечивают организацию её функций: механизмы поддержки многоуровневой системы привилегий, системы аппаратных прерываний, обеспечения многозадачности.

- *Машинно-зависимые компоненты операционной системы*. Этот слой образуют программные модули, скрывающие от вышестоящих уровней сложность аппаратного обеспечения.
- *Базовые механизмы ядра.* На этом уровне реализуются базовые функции ядра: переключение контекстов процессов, обработка прерываний, перемещение страниц из памяти на диск и обратно, и другие.
- *Менеджеры ресурсов.* На данном уровне функциональные модули реализуют наиболее важные задачи по управлению основными ресурсами компьютера. Обычно на данном уровне функционируют диспетчеры процессов, ввода-вывода, файловой системы и ОЗУ.
- *Интерфейс системных вызовов.* Представляет собой самый верхний уровень ядра, взаимодействующий непосредственно с прикладными программами и утилитами, образуя программный интерфейс операционной системы. Функции API позволяют прикладным программам получить доступ к ресурсам ПК в удобной форме без учета деталей их физического устройства.

Данное представление ядра ОС является достаточно условным. В реальных системах структура и распределение функций может незначительно отличаться.

1.2 Микроядерная архитектура

Микроядерная архитектура является противоположностью традиционной архитектуре ОС. В данной архитектуре в режиме ядра работает небольшая часть операционной системы, называемая микроядром (рисунок 3). Микроядро обычно включает машинно-зависимые модули, реализующие основные (но не все) функции по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений и управлению устройствами ввода-вывода. Множество функций микроядра сопоставимо с списком функций уровня базовых механизмов обычного ядра. Подобные функции определяют качество ОС и требуют надежной работы, что определяет необходимость использования привилегированного режима.

Остальные высокоуровневые функции ядра реализуются программами, выполняющимися в пользовательском режиме. В различных ОС по-разному решается вопрос о том, какие из системных функций должны выполняться в привилегированном режиме, а какие нет. Обычно большинство диспетчеров важных ресурсов являются частью ядра, а файловая система, подсистемы управления виртуальной памятью и процессами, менеджер безопасности работают в непривилегированном режиме.

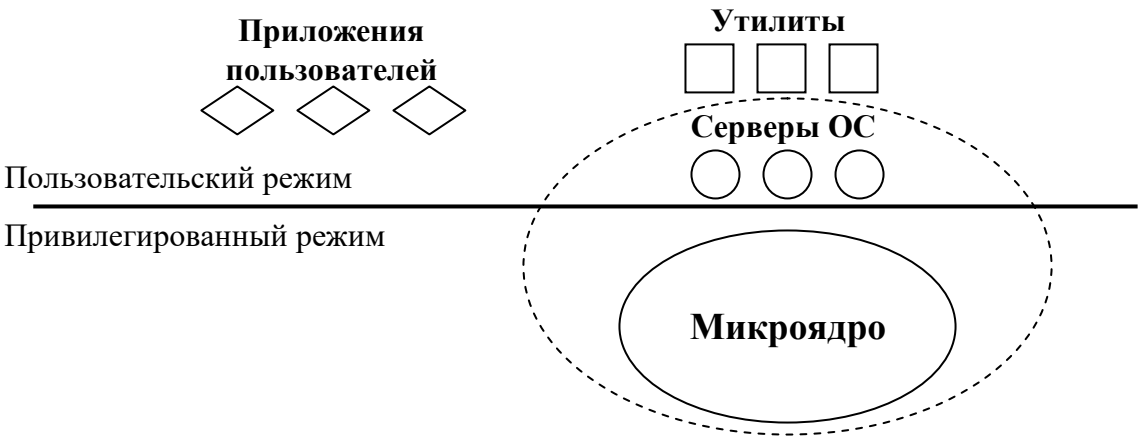


Рисунок 3 – Микроядерная архитектура

Диспетчеры ресурсов, работающие в пользовательском режиме, называются серверами ОС. Основное назначение данных модулей заключается обслуживание запросов прикладных программ и других модулей операционной системы. Для построения микроядерной архитектуры необходимо наличие в операционной системе эффективного механизма вызова процедур другого процесса.

Механизм использования функций ОС, реализованных в виде серверов, представлен на рисунке 4. Работа операционной системы данного класса соответствует модели клиент-сервер. В качестве клиентов выступают прикладные программы или компоненты операционной системы. Клиент отправляет сообщение с запросом на выполнение определенной функции у соответствующего сервера. Непосредственный обмен сообщениями между процессами невозможен вследствие изолированности их адресного пространства относительно друг друга. Поэтому в качестве посредника используется микроядро, которое имеет доступ к адресным

пространствам каждой из этих программ. Таким образом, клиент отправляет запрос, содержащий имя и параметры вызываемой функции, микроядру, который передает его серверу. После выполнения функции сервером в пользовательском режиме микроядро операционной системы передает результаты обратно клиенту с помощью другого сообщения.

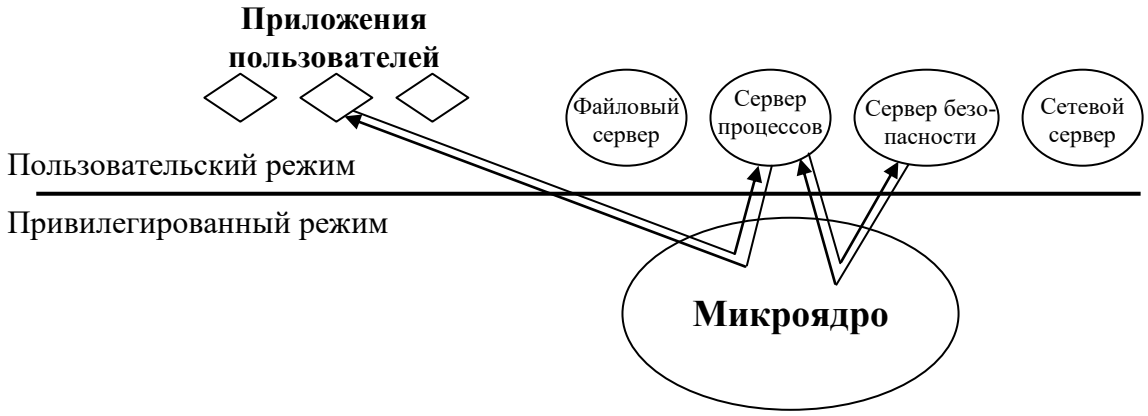


Рисунок 4 – Сема организации системного вызова в микроядерной ОС

Из рассмотренной схемы реализации системного вызова виден основной недостаток данной архитектуры ОС: при выполнении системного вызова осуществляется как минимум четыре переключения режимов – из привилегированного в пользовательский и обратно. Это приводит к снижению производительности системы вследствие наличия дополнительных служебных расходов на организацию переключений режимов.

При этом для микроядерной архитектуры характерны следующие достоинства:

- переносимость;
- расширяемость;
- конфигурируемость;
- надежность.

Переносимость ОС на другие системы определяется тем, что аппаратно-зависимый код реализован в маленьком микроядре. Для перехода на вычислительную систему с другим процессором потребуется внести изменения только в ядро операционной системы, которые в сравнении с классическими системами незначительны.

Применение клиент-серверной модели позволяет просто добавлять новый функционал в операционную систему. При этом микроядро ОС практически не изменяется. Это позволяет использовать операционные системы десятилетиями без больших изменений. Также для микроядерных операционных систем характерна простота конфигурирования параметров компьютера: достаточно изменить конфигурационные файлы или осуществить запуск/ остановку серверов системы.

Основным достоинством данной архитектуры ОС является высокая степень надежности. Во-первых, это обусловлено тем, что адресное пространство серверов изолировано друг от друга и защищено от вредоносного воздействия других модулей. Это выгодно отличает их от традиционных операционных систем, где модули ядра могут воздействовать на другие. Во-вторых, небольшой размер ядра снижает вероятность наличия ошибок в его коде.

Современные операционные системы нельзя четко отнести к классической или микроядерной архитектуре. Операционные системы образуют спектр, начиная с систем с минимальным ядром и заканчивая системами с большим объемом функций.

1.3 Операционная система Windows

Windows 2000 являлась очень удачной операционной системой, которая во многих местах используется и по сей день. Начиная с данной версии Windows, были совмещены две технологии, развивавшиеся ранее параллельно и представленные в операционных системах Windows 98 и Windows NT 4.0. С этой версии в Windows отсутствует режим MS-DOS. После выхода Windows 2000 стало традицией распространять средства для разработчиков Software Development Kit (SDK) и Driver Development Kit (DDK), которые можно найти по адресу msdn.microsoft.com. Windows 2000 представляет собой чрезвычайно сложную систему, состоящую, примерно, из 30 миллионов строк на C. А последующие ОС (XP, 2003, Vista, 7, 8, 10) содержат ещё больший объём программного кода.

Операционная система Windows реализует набор системных вызовов. Однако

взамен их непосредственного использования для программистов корпорация Microsoft предоставляет библиотечные функции интерфейсов Win32 API (Application Programming Interface) или Win64, которые либо обращаются к системным вызовам, чтобы выполнить некоторую работу, либо выполняют работу прямо в пространстве пользователя. Функции данной библиотеки опубликованы и полностью документированы. При этом сохраняется поддержка старых API функций с предыдущих версий операционной системы (Рисунок 5). Win32s – дополнительная библиотека, преобразующая подмножество 32-разрядных вызовов в 16-разрядные.

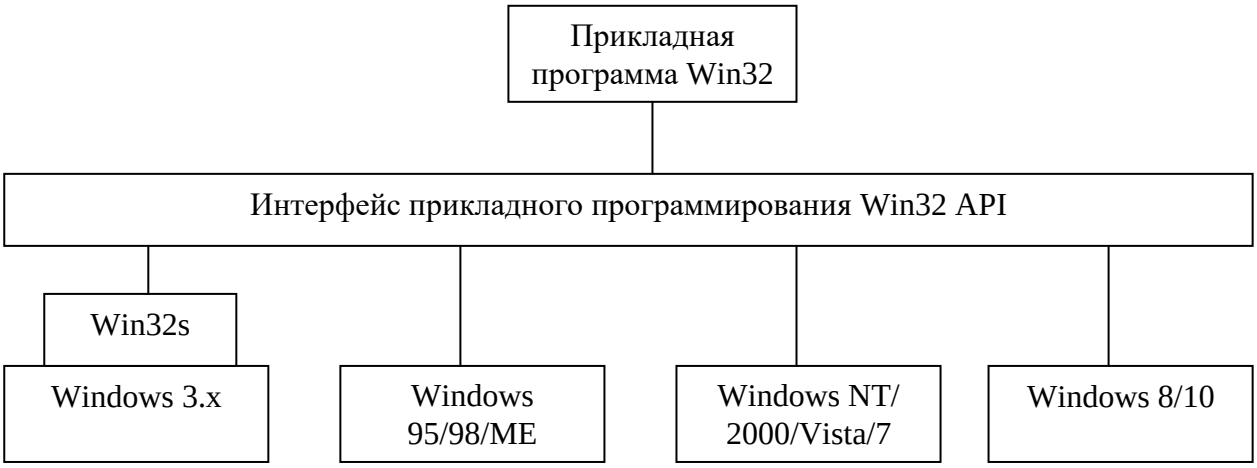


Рисунок 5 – Интерфейс Win32 API

В отличие от других операционных систем, например UNIX, Win32 API предоставляет интерфейс, позволяющий выполнить одно действие несколькими способами. Тогда как в UNIX все системные вызовы формируют минимальный интерфейс, ни один из которых нельзя удалить.

Важным понятием в операционной системе Windows является реестр – центральная база данных, в которой находится почти вся информация, необходимая для загрузки и конфигурирования ОС, настройки её под конкретного пользователя. Хотя реестр является одной из наиболее запутанных частей Windows, его идея очень проста. Он состоит из набора каталогов, каждый из которых содержит либо подкаталоги, либо записи. По своей структуре реестр напоминает файловую

систему. Информация распределена по корневым каталогам, называемым ключами. Для просмотра реестра можно пользоваться утилитой командной строки regedit. Экранная форма утилиты представлена на рисунке 6.

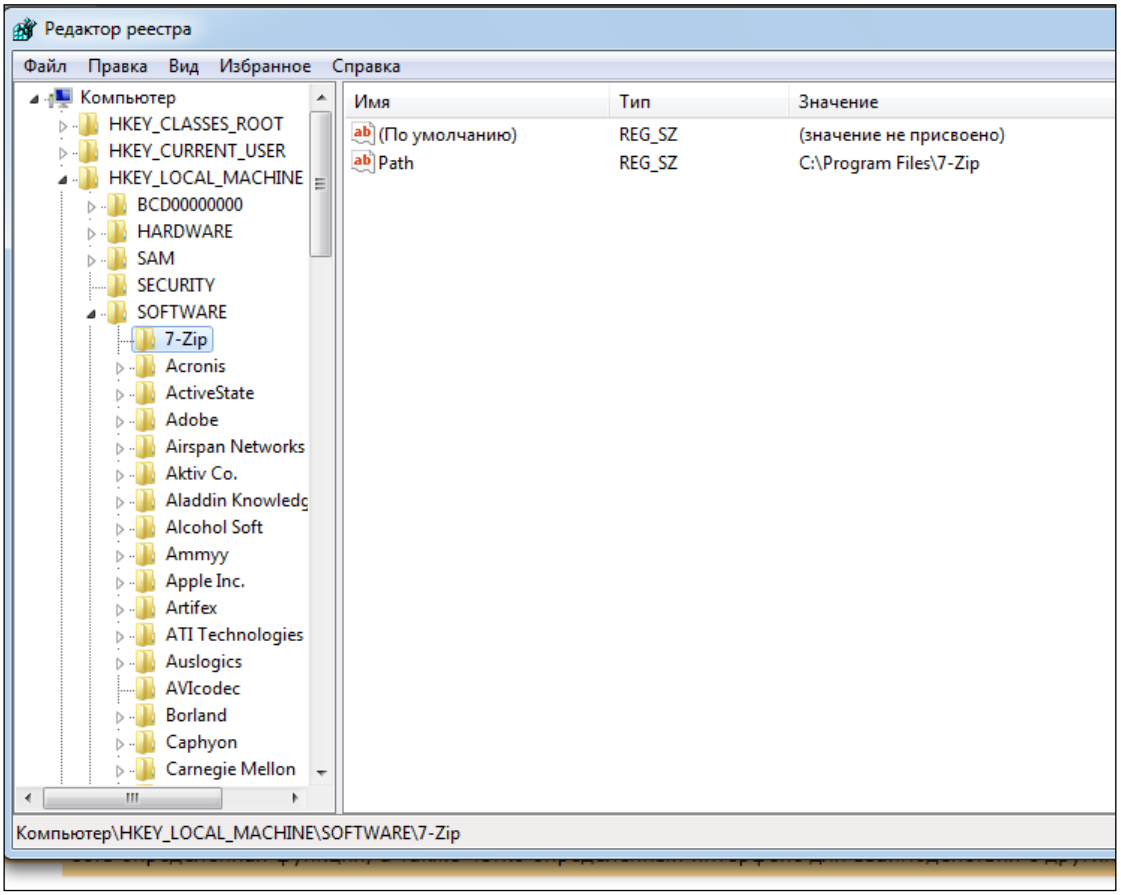


Рисунок 6- Экранная форма утилиты regedit

Ниже приведены основные разделы реестра.

HKEY_LOCAL_MACHINE – содержит всю информацию о локальной системе: описания аппаратуры, данные о драйверах, именах пользователей и паролях, политике безопасности, настройки производителей программного обеспечения для своих программ, информацию о загрузке системы.

HKEY_USERS – содержит профили для каждого пользователя.

HKEY_CLASSES_ROOT – содержит настройки для управления объектами COM (Component Object Model – модель компонентных объектов), а также занимается установкой соответствий между расширениями файлов и программами.

HKEY_CURRENT_CONFIG – представляет собой ссылку на подраздел HKEY_LOCAL_MACHINE, содержащий информацию о текущей конфигурации аппаратного обеспечения.

HKEY_CURRENT_USER – указывает на настройки текущего пользователя.

HKEY_PERFORMANCE_DATA – данный ключ не виден в утилите просмотра regedit. Операционная система содержит сотни счетчиков для мониторинга производительности системы. К таким счетчикам можно получить доступ через этот ключ реестра.

Операционная система Windows состоит из двух основных частей: самой операционной системы, работающей в режиме ядра, и подсистем окружения, работающих в режиме пользователя (рисунок 7) [20]. Ядро является традиционным в том смысле, что оно управляет процессами, памятью, файловой системой и т. д. Подсистемы окружения являются отдельными процессами, помогающими пользователю выполнять определенные системные функции.

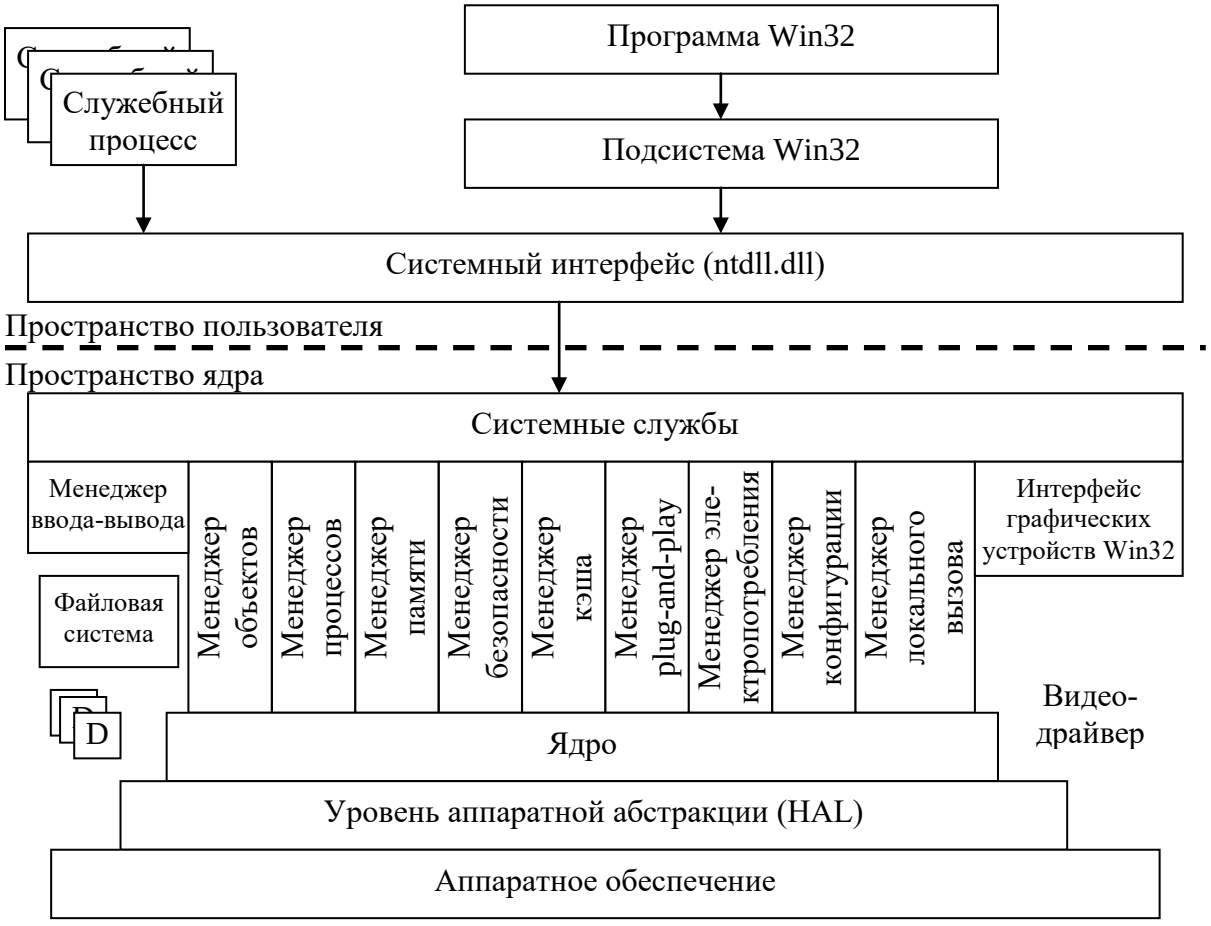


Рисунок 7 – Структура операционной системы 32-битной Windows

Система разделена на несколько уровней, каждый из которых пользуется службами нижнего уровня. Эта структура проиллюстрирована на рисунке 7 (квадратики, помеченные символом «D», обозначают драйверы устройств, а

сервисные процессы являются системными демонами). Кратко опишем данную схему.

Уровень аппаратных абстракций HAL (Hardware Abstraction Layer) предназначен для скрытия особенностей и деталей аппаратных средств компьютера. Данный уровень предоставляет вычислительной системе абстрактные аппаратные устройства вместо разнообразного реального аппаратного обеспечения. Данные абстрактные устройства реализуются в виде аппаратно-независимых служб (процедурных вызовов и макросов) и используются остальной частью ОС и драйверами. Так как драйверы и ядро операционной системы пользуются службами HAL (идентичными на всех операционных системах Windows, независимо от аппаратного обеспечения) и не обращаются напрямую к устройствам, то данное обстоятельство позволяет упростить процесс переноса на другую платформу. Перенос самого уровня HAL относительно прост, так как весь машинно-зависимый код сконцентрирован в одном месте, а цель переделки четко определена, то есть заключается в реализации всех служб уровня HAL.

Примерами функций, которые реализуются службами уровня HAL, являются: доступ к регистрам устройств, обработка прерываний, операции DMA (Direct Memory Access – прямой доступ к памяти), управление таймерами, часами реального времени, спин-блокировками нижнего уровня, синхронизация многопроцессорных конфигураций, интерфейс с BIOS и доступ к CMOS-памяти. Для специфических устройств ввода-вывода, например, клавиатур, мышей и дисков, блоков управления памятью (MMU) уровень HAL не реализует абстракций.

Уровень HAL зависит от конкретного аппаратного обеспечения. Поэтому операционная система Windows 2000 поставляется на компакт-диске с различными версиями HAL под конкретные платформы. При установке операционной системы определяется необходимый уровень и копируется в системную папку под именем файла hal.dll.

Эффективности уровня HAL для мультимедийных программ может быть недостаточно. Поэтому фирма Microsoft разработала набор библиотек, называемый DirectX, позволяющий расширить функциональность уровня HAL и

предоставляющий пользовательским программам возможность прямого доступа к аппаратным средствам.

На следующем уровне располагаются ядро и драйверы устройств. Ядро ОС позволяет сделать остальную часть операционной системы независимой от аппаратных средств и легко переносимой на другие платформы. При этом ядро получает доступ к аппаратным средствам через уровень HAL.

Ядро реализует абстрактную модель аппаратного обеспечения более высокого уровня, управляет переключениями потоков, предоставляет низкоуровневую поддержку управляющим объектам и объектам диспетчеризации. Таким образом, ядро реализует базовые примитивы, которые используются на верхнем уровне исполняющей системой. В частности, объекты ядра используются для создания дескрипторов исполняющих объектов.

Управляющие объекты уровня ядра – это объекты, управляющие системой, включая примитивные объекты процессов, объекты прерываний. Также в данную группу объектов относятся объекты отложенного вызова процедур DPC и асинхронного вызова процедур APC. DPC используется, чтобы отделить часть процедуры обработки прерываний, для которой время является критичным, от той ее части, для которой время некритично. APC похож на отложенный вызов процедуры DPC, только выполняется в контексте определенного процесса. Для выполнения задач синхронизации и планирования на уровне ядра используются семафоры, мьютексы, события, таймеры и другие объекты.

На базе ядра и драйвера устройств реализуется исполняющая система (супервизор или диспетчер). Данная система состоит из 10 компонентов, представляющих собой набор процедур, использующихся совместно для выполнения определенной задачи. Необходимо отметить, что компоненты данного уровня могут вызывать друг друга.

Менеджер объектов управляет всеми объектами ОС: процессами, потоками, файлами, каталогами, семафорами, устройствами ввода-вывода, таймерами и многими другими. *Менеджер ввода-вывода используется для* формирования каркаса для управления устройствами ввода-вывода и предоставления общих служб ввода-

вывода. Он предоставляет остальной части системы независимый от устройств ввод-вывод, вызывая для выполнения физического ввода-вывода соответствующий драйвер. Управление процессами и потоками, включая их создание и завершение, осуществляет *менеджер процессов*. *Менеджер памяти* реализует архитектуру виртуальной памяти со страничной подкачкой по требованию операционной системы Windows. Он управляет преобразованием виртуальных страниц в физические страничные блоки. *Менеджер безопасности* приводит в исполнение сложный механизм безопасности Windows, удовлетворяющий требованиям класса C2 Оранжевой книги Министерства обороны США. *Менеджер кэша* хранит в памяти блоки диска, которые использовались в последнее время, для ускорения доступа к ним. Его работа состоит в том, чтобы определить какие блоки понадобятся снова в ближайшее время. *Менеджер plug-and-play* получает все уведомления об установленных новых устройствах. *Менеджер энергопотребления* позволяет отключить монитор и диски, если к ним не было обращений в течение определенного интервала времени для снижения потребления электроэнергии. *Менеджер конфигурации* отвечает за состояние реестра. *Менеджер вызова локальной процедуры* обеспечивает высокоэффективное взаимодействие между процессами и их подсистемами.

Интерфейс графических устройств GDI (Graphic Device Interface) занимается управлением графическими изображениями для различных устройств вывода информации (мониторы, принтеры). Данный интерфейс реализует системные вызовы, позволяющие пользовательским программам выводить данные независимо от типа устройства вывода.

Тонкий уровень системных служб располагается над исполняющей системой. Он предназначен для предоставления интерфейса к исполняющей системе. Данный уровень принимает системные вызовы Windows и вызывает соответствующие модули исполняющей системы для их выполнения.

1.4 Операционная система Linux

История Linux и Unix хорошо описана в классическом труде Э. Таненбаума [20]. Можно выделить несколько этапов развития ОС Linux. Но нужно понимать, что речь идет прежде всего об истории ядра ОС Linux, а не развитии дистрибутивов операционных систем.

1 этап. Создание ОС Linux. Первая версия возникла в 1991 году. Фактически это была переделка ОС Minix, которую для учебных целей разработал Э. Таненбаум. Размер исходного текста составил 9300 строк на языке С и 950 строк на ассемблере. После этого операционная система быстро развивалась и росла в размерах.

2 этап. Версия 1.0. 1994 год. Первая полноценная версия ОС Linux. Состояла примерно из 165 000 строк кода и включала новую файловую систему, реализовывала отображение файлов на адресное пространство памяти и совместимое с BSD сетевое программное обеспечение с сокетами и TCP/IP . Она также включала многие новые драйверы устройств [2].

3 этап. Версия 2.0. С 1996 года – по настоящее время. Версия состояла примерно из 470 000 строк на языке С и 8000 строк ассемблерного кода. В операционную систему уже были добавлены поддержка 64-разрядной архитектуры, симметричная многозадачность, новые сетевые протоколы и много новых функций. Причем большую часть ОС составляла обширная коллекция драйверов устройств для постоянно растущего количества поддерживаемых периферийных устройств[14]. Несмотря на то, что выпущена уже версия 4.17 кардинально новых изменений в структуре ядра не появилось.

Особенностью Linux является то, что это бесплатное программное обеспечение, которое распространяется на основе GPL (GNU Public License — общедоступная лицензия GNU). Данная лицензия позволяет пользователям бесплатно использовать, копировать, модифицировать и распространять исходные коды и двоичные файлы. При продаже двоичного кода Linux должно одновременно или по требованию предоставляться исходный код.

По сути этим и притягательна ОС Linux. Поскольку она и большое количество

программ, которые работают с данной ОС, распространяются с открытыми исходными кодами под определенным видом лицензии. Основные виды текущих лицензий с открытым исходным кодом:

- BSD (Berkeley Software Distribution license);
- GNU GPL (GNU General Public License разных версий);
- GNU LGPL (GNU Lesser General Public License);
- Apache 2.0;
- и другие (MIT, EPL, MPL и т.п.).

Лицензия типа BSD является самой лояльной, позволяя практически неограниченно использовать исходный код программного обеспечения в производных продуктах, в том числе и коммерческих. И выпускать этот продукт под другой лицензией.

GNU GPL позволяет модифицировать, копировать и распространять программное обеспечение, в то же время гарантируя, что права автора будут сохранены. Однако модификация программного обеспечения возможно только с сохранением вида лицензии и указанием авторства изначального продукта. Это предполагает, что исходный код всего продукта на базе продукта под лицензией GNU GPL должен быть открытым. Однако это не запрещает продавать этот новый продукт производителями.

GNU LGPL – облегченная версия GNU GPL. Ее отличия заключаются в том, что не нужно производные продукты выпускать под той же лицензией, а можно создать его путем компоновки из различных модулей. И если один из этих модулей (например, Dll в ОС Windows, или so-файл в Linux) – это модуль под лицензией LGPL. То обычно надо просто предоставить исходный код этого модуля вместе с продуктом или указание на то, как этот исходный код получить и скомпилировать конечный модуль.

Apache 2.0 – также очень распространенная разрешительная лицензия (это означает, что она больше разрешает, чем запрещает). Можно создавать новый продукт с другой лицензией на базе решений под Apache 2.0, но при этом нужно скопировать содержимое файла NOTICE, который обязателен для лицензии Apache

2.0, таким образом, чтобы пользователь легко мог его прочитать, например, в окно About программы. Лицензия MIT похожа на Apache 2.0.

Это наиболее распространенные виды лицензий. Все лицензии с открытым кодом легко найти в в сети Интернет, многие из них переведены на русский язык.

Когда говорят Linux, то, прежде всего, подразумевают ядро операционной системы, поскольку операционных систем на основе ядра Linux много и они называются дистрибутивами. Это Debian, Mint, Ubuntu, Arch Linux, Chakra, Manjaro, RedHat, Fedora, Mageia, OpenSUSE, Slackware, Gentoo и другие.

Как было написано в параграфе 1.1, любая ОС должна предоставлять возможности по управлению ресурсами (и это возлагается на ядро Linux) и интерфейс взаимодействия с пользователем. Как раз интерфейс взаимодействия с пользователем, настройка и администрирование, инсталляция программ – это то, чем отличаются всевозможные дистрибутивы.

Технически каждый может разработать собственный дистрибутив. Существует целый проект http://www.linuxfromscratch.org/, который посвящен вопросам разработки дистрибутивов.

Важным критерием выбора дистрибутива является соответствие его возможностей требованиям решаемой задачи. Эти данные можно получить из сайтов разработчика дистрибутива и форумов. Часто пользователи Linux операционных систем привыкают к какому-то одному дистрибутиву, например, Debian, и пытаются использовать его для решения всех задач.

Нужно учитывать, что когда говорят Linux, то имеют в виду в первую очередь ядро ОС, а GNU/Linux – это операционная система целиком. Основой ОС GNU/Linux является монолитное ядро, больше тяготеющее к классической архитектуре. В простейшей форме архитектура Linux представлена на рисунке 8.

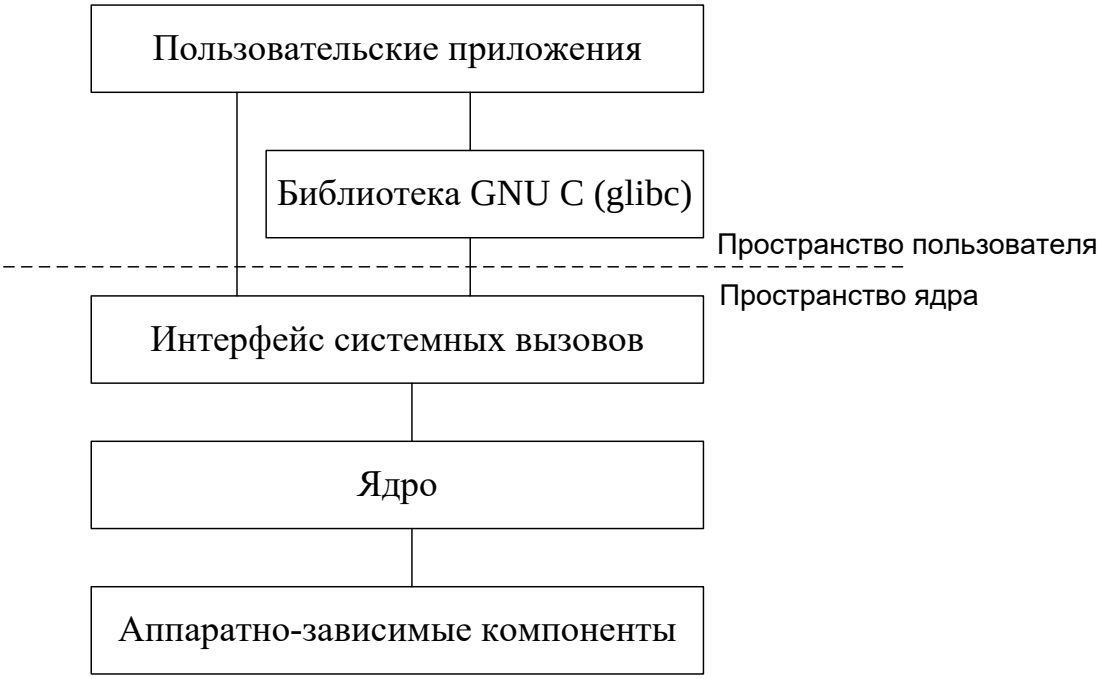


Рисунок 8 – Упрощенная структура GNU Linux

Ядро Linux поставляется в двух вариантах: стабильном (stable) и разрабатываемом (development). Версии стабильного ядра — это выпуски продукции промышленного уровня, которая готова для широкого использования. Новые стабильные версии ядра обычно выпускаются для исправления ошибок и предоставления новых драйверов устройств. Разрабатываемые версии ядра, наоборот, подвержены быстрым изменениям. По мере того, как разработчики экспериментируют с новыми решениями, в исходный код ядра довольно часто вносятся радикальные изменения.

Ядра Linux стабильных и разрабатываемых версий можно отличить друг от друга с помощью простой схемы нумерации (рисунок 9).

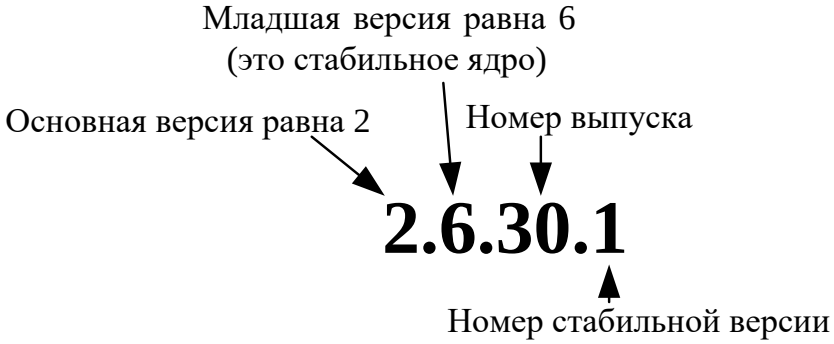


Рисунок 9 – Соглашение о нумерации ядра Linux

Версию ядра определяют три или четыре числа, которые разделяются точкой. Первое число определяет номер основной (major) версии, второе — номер младшей (minor) версии, а третье число указывает на номер выпуска (revision). Необязательное четвертое число определяет номер стабильной версии. Значение младшей версии также определяет, является ли ядро стабильным или разрабатываемым; если это значение четное, значит, ядро стабильное, а если нечетное — разрабатываемое. Так, например, версия 2.6.30.1 определяет стабильное ядро. Ядро имеет основную версию 2, младшую версию 6, номер выпуска 30 и первую стабильную версию. Первые два числа также определяют серию ядер, в данном случае серия ядер — 2.6 [9].

Если вы начинаете разрабатывать код ядра Linux, вы становитесь частью глобального сообщества разработчиков. Для того необходимы знания об устройстве операционной системы Linux.

На рисунке 10 приведена структура ядра Linux, взятая из работы [20]. Ядро работает с аппаратным обеспечением. Обработкой прерываний и синхронизацией занимается нижний уровень ядра.

За взаимодействие с аппаратными средствами компьютера, в том числе передачу данных по сети и обмен данными с внешними устройствами отвечает компонент ввода-вывода. Верхним уровнем компонента ввода-вывода является виртуальная файловая система (VFS), который интегрирует в себе все операции ввода-вывода системы. VFS является уровнем абстракции, который скрывает детали реализации аппаратных средств. Непосредственное взаимодействие виртуальной файловой системы с аппаратными средствами осуществляется через их драйверы.

Важнейшими элементами ядра Linux являются компоненты управления памятью и процессором. Управление памятью подразумевает отображение виртуальной памяти на физическую, поддержка кэша страниц. Компонент управления процессами отвечает за создание и завершение процессов. Также данный элемент осуществляет планирование работы процессов и потоков. Ядро Linux работает с процессами и потоками как с исполняемыми модулями и планирует их работу на основе глобальной стратегии планирования. Обработка сигналов,

позволяющих реализовать межпроцессное взаимодействие, также производится в данном компоненте [20].

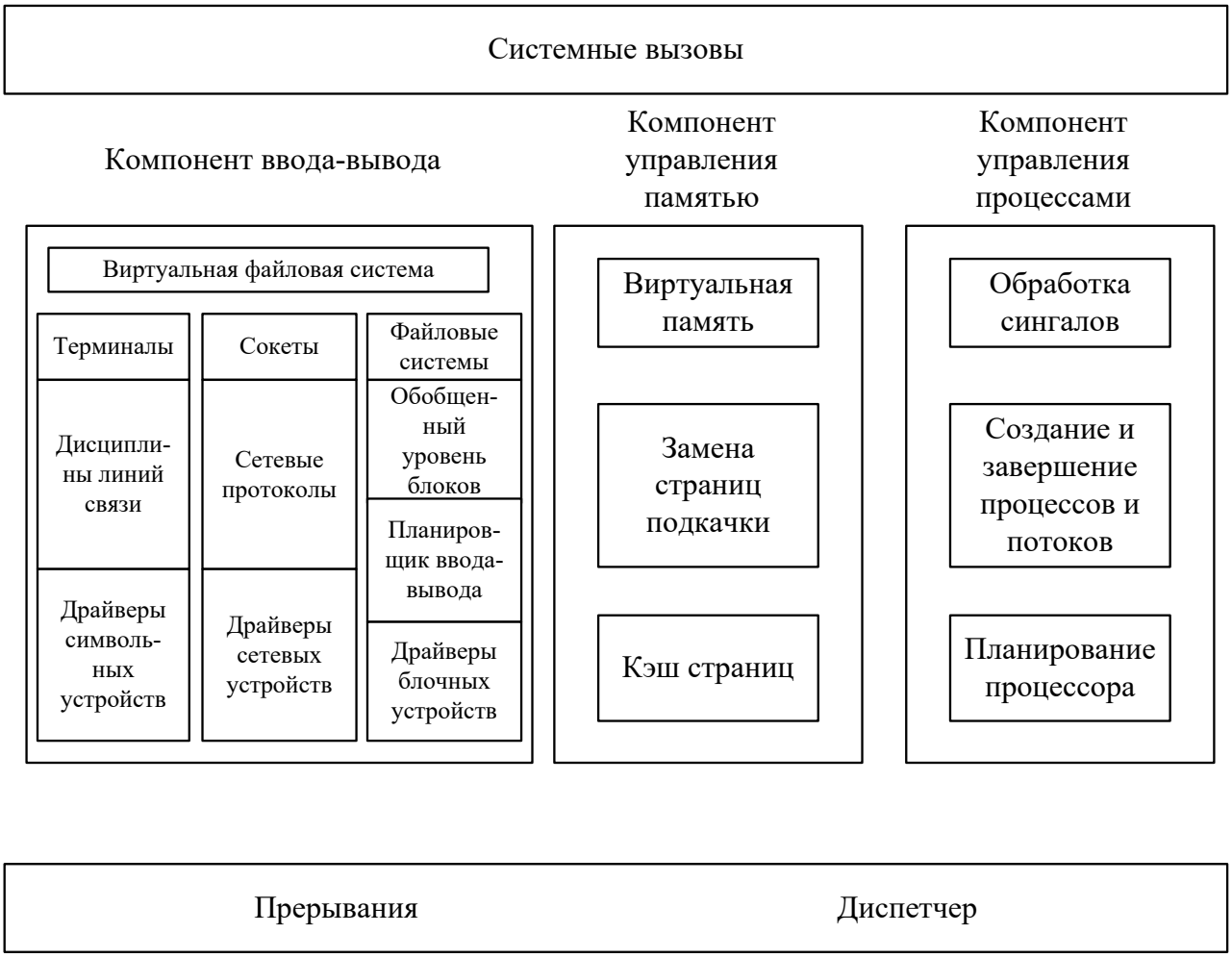


Рисунок 10 – Структура ядра Linux

На самом верхнем уровне находится интерфейс системных вызовов, позволяющий пользовательским приложениям запросить выполнение функций ядра ОС.

1.5 Командная строка Linux

Важнейшей особенностью Linux является наличие командной строки. Это не значит, что в Windows ее нет. Однако в ОС Linux часто многие настройки и использование её функций проще осуществить именно через интерфейс командной строки, который еще называется оболочкой. Существуют различные варианты оболочек, наиболее распространенной из которых является bash. Окно оболочки в графическом интерфейсе называется терминалом.

При работе с командной строкой следует обращать внимания на символ приглашения к вводу: если он $, то это означает, что пользователь работает в обычном режиме от имени пользователя. Если же строка приглашения заканчивается #, то это означает, что команды будут исполняться от имени суперпользователя (другое название – root или администратор).

В стандарте POSIX 1003.1-2008 описано более 150 программ, выступающих в качестве команд для оболочки. А в реальной операционной системе их и того больше. Ниже представлена информация о некоторых основных командах Linux.

Команда man форматирует и отображает страницы электронного справочника. Пример вызова справки о команде ls имеет следующий вид:

$man ls.

Программа ls сначала выводит список всех файлов, находящихся в каталогах, перечисленных в командной строке. Если не указано ни одного каталога, то по умолчанию используется текущая директория. Опция -d заставляет ls не считать аргументы-каталоги каталогами. В этом случае будет выводиться только информация об указанных каталогах без вывода списка их файлов. Синтаксис вызова имеет следующий вид:

ls [опции] [файл...].

С подробными опциями вы можете ознакомиться, вызвав справку командой man ls. Следующая команда выводит в командную строку список всех файлов текущего каталога с подробной информацией о них:

$ls –la.

Команда pwd предназначена для вывода информации о текущем каталоге.

Смена текущего каталога осуществляется с помощью команды cd, которой в качестве аргумента можно передавать относительный или абсолютный путь до файла. Можно указать две точки в качестве выхода на уровень каталога выше: «$cd ..».

Команда cp предназначена для копирования файлов и каталогов. Команда имеет следующий формат:

cp [опции] файл путь.

Команда cp копирует файлы. Вы можете либо копировать один файл в другой заданный файл, либо копировать сколько несколько файлов в заданный каталог. Если последний аргумент является существующим каталогом, то cp копирует каждый исходный файл в этот каталог, сохраняя их имена. В противном случае, если заданы только имена двух файлов, то cp копирует первый файл во второй. Если задано более двух аргументов, которые не являются опциями, а последний аргумент не является именем каталога, то это приведет к ошибке.

Команда mv перемещает (переименовывает) файлы. Формат команды:

mv [опции...] исходный_файл файл_назначения

mv [опции...] исходный_файл... каталог

Если последний аргумент является именем существующего каталога, то mv перемещает все остальные файлы в этот каталог. В противном случае, если задано только два файла, то имя первого файла будет изменено на имя второго. Если последний аргумент не является каталогом и задано более чем два файла, то будет выдано сообщение об ошибке.

При операции с файлами (cp, mv и другие файлы) можно использовать служебные символы ? и *. ? обозначает один случайный символ, * - неопределенное количество (или ноль) случайных символов. Они используются для манипуляции с группами файлов по определенным критериям.

Команда rm удаляет файлы или каталоги. Формат:

rm [опции] файл...

Для удаления каталогов используется похожая команда rmdir.

Команда mkdir создает новый каталог с именем a:

$ mkdir a

Для создания ссылки на файл используется команда ln. Формат:

ln [-f] файл1 [файл2 ...] целевой_файл

Команда ln делает целевой_файл ссылкой на файл1. Файл1 не должен совпадать с целевым_файлом.

Для вывода информации в стандартный поток информации используется команда echo.

Стандартный поток информации означает консоль, однако эту информацию можно перенаправить, например, в файл с помощью знака >.

Следующая информацию выводит файлы текущего каталога в файл с именем file:

$ls -la > file

Команд довольно много и с ними вы можете ознакомиться в документации.

1.6 Контрольные вопросы

1. Дайте определение операционной системы.
2. Назовите основные функции операционной системы.
3. Что такое системный вызов?
4. Какие задачи решает ОС при управлении ресурсами?
5. Какими двумя основными способами реализуется управление ресурсами?
6. Дайте характеристику операционных систем мэйнфреймов?
7. Какой класс операционных систем обрабатывает запросы за жестко заданный интервал времени?
8. Дайте характеристику ОС классической архитектуры.
9. Зачем операционная система должна работать в привилегированном режиме?
10. В привилегированном или пользовательском режиме работают утилиты в ОС классической архитектуры?
11. В чем недостаток использования механизма привилегий при исполнении системных вызовов?
12. Определите достоинства и недостатки микроядерной архитектуры.
13. Что такое Win32 API?
14. В чем назначение реестра?
15. В чем назначение уровня аппаратных абстракций ОС Windows?
16. В чем назначение интерфейса графических устройств?
17. Какими особенностями обладает тип лицензии GNU/GPL?

2 Процессы и потоки

2.1 Процессы

Большинство современных операционных систем относятся к классу многозадачных, позволяющих исполнять несколько программ одновременно, по крайне мере создавая такое представление для пользователя. В такой системе процессор переключается между программами, предоставляя каждой от десятков до сотен миллисекунд времени (квант времени). В каждый конкретный момент времени процессор работает только с одной программой, создавая иллюзию параллельной работы, реализуя псевдопараллелизм [20]. Реальная параллельная работа программ может осуществляться в многопроцессорных и многоядерных системах. Например, в четырехъядерной системе могут выполняться одновременно 4 процесса. Так как количество запущенных программ гораздо больше, то операционная система вынуждена также реализовывать псевдопараллизм.

Для изучения работы параллельно работающих программ разработчики ОС создали концептуальную модель последовательных процессов. В данной модели все работающие программы представляются в виде набора последовательных процессов. При этом для каждого процесса создается собственный виртуальный процессор.

В качестве примера рассмотрим ситуацию, когда запущены 4 программы на однопроцессорной системе. На рисунке 11, а представлена схема распределения памяти между работающими 4 процессами. Процессор переключается от выполнения команд одного процесса на команды другого. При этом он использует один физический счетчик команд. Результаты приведения к концептуальной модели представлены на рисунке 11, б , в котором представлены 4 независимых процесса, каждый со своим логическим счетчиком команд. Несмотря на то, что концептуальная модель предполагает наличие независимых процессов, на самом деле для однопроцессорной системы в каждый момент выполняется только один (рисунок 11, в).

Порядок переключения процессора между процессами осуществляется под управлением операционной системы и зависит от текущего состояния вычислительной машины. При каждом запуске программы его временная диаграмма исполнения будет отличаться. Поэтому разработка приложений должна производится без жестких предположений о его времени выполнения.

Существует следующие основные способы создания процессов:

- инициализация системы;
- осуществление из текущего процесса системного запроса на создание другого процесса;
- запрос пользователя на создание процесса;
- инициирование пакетного задания.

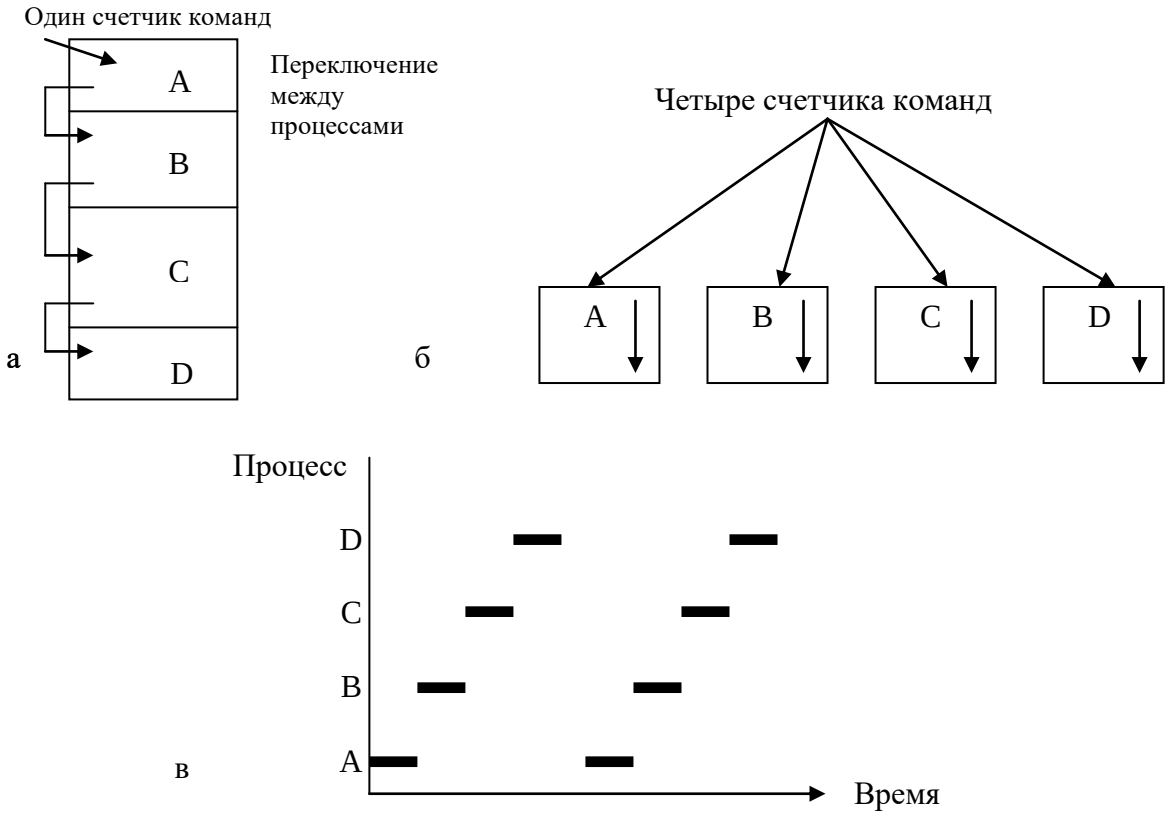


Рисунок 11 – 4 программы в многозадачном режиме (а); модель 4 независимых последовательных процессов (б); в каждый момент времени активна только одна программа (в)

В коде программы для создания другого процесса в UNIX-системах должна

вызваться комбинация функций fork() и exec(), а в операционной системе Windows – CreateProcess [18].

Завершение работы процесса реализуется следующими основными способами:

- обычный выход (преднамеренно);
- выход по ошибке (преднамеренно);
- выход по неисправимой ошибке (непреднамеренно);
- уничтожение другим процессом (непреднамеренно).

В программном коде для завершения процесса в UNIX должен вызваться системный запрос kill, которому в ОС Windows соответствует функция TerminateProcess.

В операционных системах Windows и UNIX по-разному реализуется связь между родительским и дочерним процессами. В UNIX реализована иерархия процессов, а в Windows все процессы равноправны. Аналогом механизма организации иерархии процессов в Windows является возможность получить родительскому процессу специальный маркер (так называемый дескриптор), который позволяет контролировать дочерний. Данная иерархия может быть нарушена в том случае, когда маркер передан другому процессу.

Основными состояниями процесса являются следующие (рисунок 12):

- действие (в конкретный момент времени используется процессор);
- готов к работе (у процесса есть все ресурсы, кроме времени процессора);
- блокировка (процесс приостановлен в ожидании внешнего события, например, системного вызова ввода данных).

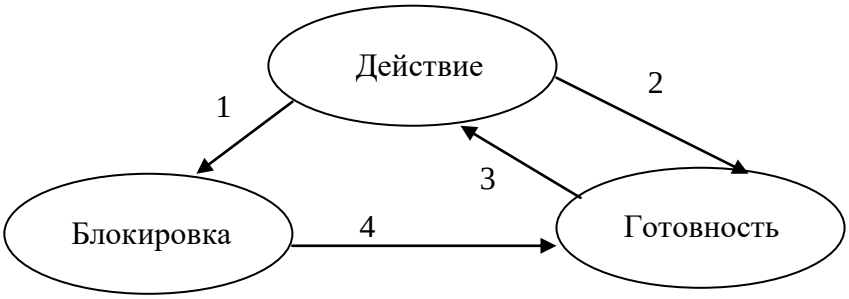


Рисунок 12 – Схема возможных состояний процесса

Переход 1 осуществляется, когда в коде встречается команда ввода данных, которая не может быть реализована быстро. Чтобы не тратить время процессора на ожидание завершения системного вызова ввода-вывода данных, планировщик ОС незамедлительно переключает процессор на другой процесс, а текущий помечается как заблокированный и выводится из очереди для получения времени процессора. Операционная система переводит процесс из состояния «Действие» в «Готовность», когда истек квант (интервал) времени, выделенный процессу. Все процессы с состоянием «Готовность» находятся в очереди к процессору. ОС с учетом выбранного алгоритма планирования выбирает один из процессов с состоянием «Готовность», переводит обратно в «Действие» и предоставляет ему квант времени процессора.

Переходы 2 и 3 осуществляются планировщиком ОС таким образом, что процессы даже не знают о существовании этих переключений. Переход из состояния «Блокировка» в «Готовность» происходит при возникновении ожидаемого события, например, получения входных данных.

Для управления процессами ОС формирует таблицу (массив структур) процессов, каждый элемент (дескриптор) которого характеризует определенный процесс. Элемент таблицы процессов содержит следующую информацию:

- состояние процесса;
- счетчики команд;
- указатели стека;
- распределение памяти;
- данные о выделенных ресурсах, например, открытых файлах;
- информация, которая используется при переключении процессов.

В таблице 1 представлены некоторые типичные поля дескриптора процесса.

Таблица 1 – Основные поля дескриптора процесса

Управление процессом	**Управление памятью**	**Управление файлами**
Регистры Счетчик команд Слово состояния программы Указатель стека Состояние процесса Приоритет Параметры планирования Идентификатор процесса Родительский процесс Группа процесса Сигналы Время начала процесса Использованное процессорное время Процессорное время дочернего процесса	Указатель на текстовый сегмент Указатель на сегмент данных Указатель на сегмент стека	Корневой каталог Рабочий каталог Дескрипторы файла Идентификатор пользователя Идентификатор группы

Реализация многопоточности в системах с вытесняющей многозадачностью базируется на прерываниях от таймера. Прерывание (англ. interrupt) – сигнал, сообщающий процессору о совершении какого-либо асинхронного события [20]. При этом процессором приостанавливается выполнение текущей последовательности команд, и управление передаётся подпрограмме обработке прерывания (обработчику прерывания), которая осуществляет обработку события и возвращает управление в прерванный код. Прерывание от таймера операционной системой используется для отсчитывания интервала времени, выделенного для выполнения определенного процесса.

Программа и процесс представляют собой разные понятия. Программа представляет собой статический набор команд, а процесс - это набор ресурсов, кода и данных, использующихся при выполнении программы. Таким образом, процесс – это объект исполнения, использующийся ОС для управления запущенной

программой. Основными компонентами процесса являются:

- служебные данные о процессе, использующиеся операционной системой для управления им и планирования;
- диапазон адресов виртуальной памяти, выделенный запущенной программе;
- код и данные программы, проецируемые на виртуальное адресное пространство процесса.

2.2 Потоки

Для управления запущенными программами операционная система использует два объекта исполнения: процесс и поток. Процесс представляет собой пассивный объект, контейнер, который определяет выделенные ресурсы приложению. В свою очередь, поток (или нить) – это независимый путь выполнения команд внутри процесса, разделяющий вместе с процессом общее адресное пространство, код и глобальные данные. Кроме того, поток содержит собственные регистры, стек и механизмы ввода, в том числе очередь скрытых сообщений. Каждый процесс содержит по крайне мере один поток.

Широкое применение многоядерных процессоров в компьютерах определяет необходимость использования методов параллельного решения задач для сокращения их времени выполнения. Это может быть реализовано двумя основными способами: использование многопоточных приложений или совокупности взаимодействующих процессов. Как показано в таблице 2, потоки разделяют не только адресное пространство, но и открытые файлы, дочерние процессы, сигналы и т. п. , что позволяет эффективно им взаимодействовать.

В первом столбце таблицы представлены элементы процесса, являющиеся общими для всех его потоков. Например, если один поток открывает файл, то он становится доступным для всех остальных. Также как и процесс, поток может находиться в одном из нескольких состояний. Переходы между состояниями потоков такие же, как на рисунке 12.

При этом у каждого потока есть индивидуальные элементы, например, свой собственный стек, выделенный в адресном пространстве процесса. Стек (англ. stack – стопка) – структура данных с методом доступа к элементам LIFO (англ. Last In – First Out, «последним пришел – первым вышел») [20]. Каждый поток характеризуется аппаратным контекстом, в упрощенном виде представляющим собой копию регистров процессора. Это связано с реализацией многозадочности: процессор фактически переключается не между процессами, а потоками, которые могут принадлежать разным процессам. Так как обработка данных непосредственно производится с использованием регистров процессоров, то при переключении потока нужно сохранить текущее результаты, чтобы потом можно было продолжить вычисления. На рисунке 13 представлена структура многопоточного приложения, состоящая из трех потоков.

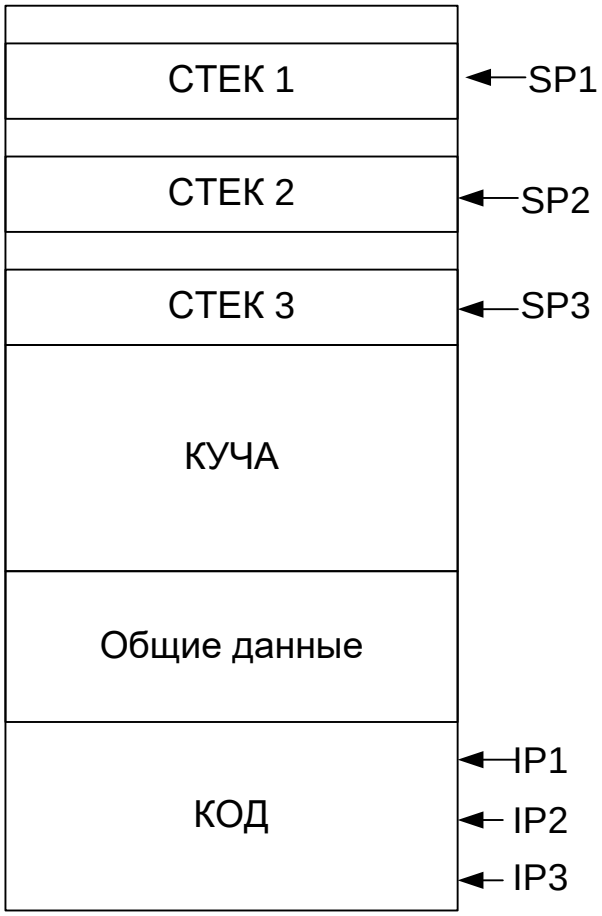


Рисунок 13 – Структура многопоточного приложения

В качестве примера использования нескольких потоков в одном процессе,

можно привести ситуацию, когда приложению нужно записать большой файл на диск. При использовании одного потока доступ к другим функциям программы будет недоступен до окончания операции записи в файл. Другой пример – текстовый редактор, где один поток отвечает за ввод и отображение текста, а другой – за проверку орфографии. Также многопоточные приложения используются для снижения времени решения задачи, например, архиватор 7zip для сжатия файлов.

Таблица 2 –Общие и индивидуальные элементы потоков

Элементы процесса	**Элементы потока**
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и их обработчики	
Информация об использовании ресурсов	

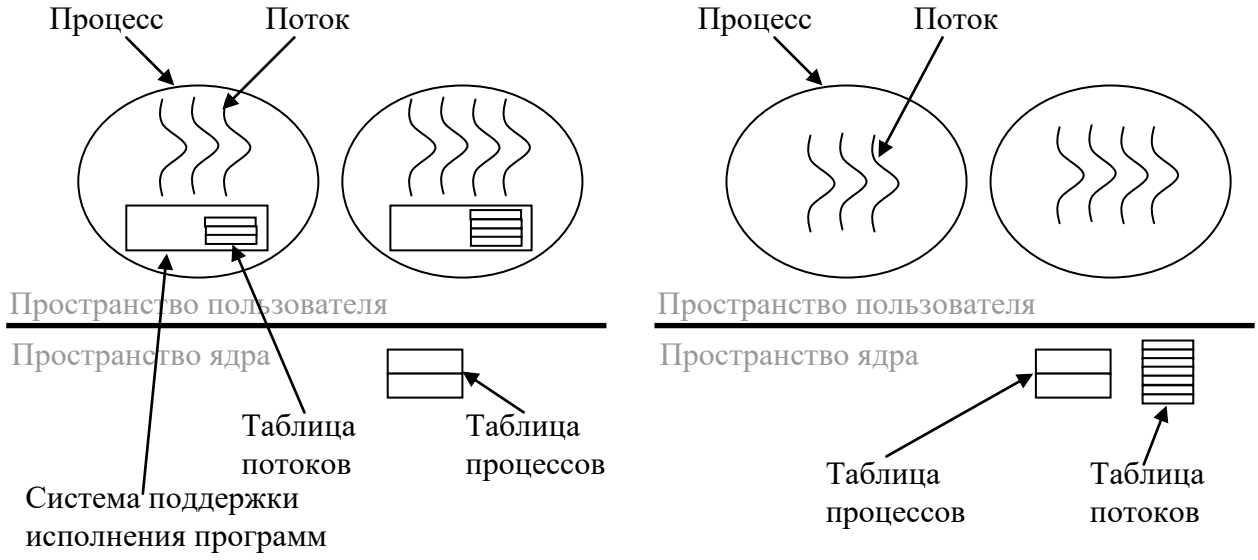
Преимуществами применения многопоточных приложений перед множеством взаимодействующих процессов являются:

- возможность совместного использования потоками адресного пространства и всех содержащихся в нём данных. Сокращается время обмена данными между параллельными элементами, так как взаимодействие осуществляется через глобальные переменные в оперативной памяти;
- создание и уничтожение потоков происходит в примерно в 100 раз быстрее, чем для процессов;
- увеличивается производительность.

К недостатку многопоточных приложений можно отнести плохую масштабируемость, так как не позволяет использовать возможности нескольких компьютеров, например, кластеров.

Существует два основных способа реализации пакета потоков: в пространстве пользователя и средствами ОС на уровне ядра (рисунок 14). В первом случае, поток,

реализованный ядром ОС, разбивается на подпотоки средствами пользовательской программы. При этом операционная система ничего не знает о пользовательских потоках и управляет обычным однопоточным процессом. Достоинством данного подхода является то, что его можно реализовать даже в ОС, не поддерживающих многопоточность. Также обеспечивается более высокая производительность по отношению ко второму способу, так как создание потоков на уровне ядра требует больше ресурсов. При этом присутствует возможность использовать собственный алгоритм планирования работы потоков.



а б

Рисунок 14 – Пакет потоков в пространстве пользователя (а); пакет потоков, управляемый ядром (б)

Основным недостатком данного способа является то, что операционная система не видит потоков пользовательского уровня и предоставляет им время процессора как одному потоку. Также при блокировке потока ядра будут заблокированы и пользовательские. В большинстве известных операционных системах потоки реализуются в ядре или применяется гибридная схема.

2.3 Межпроцессное взаимодействие

Для эффективного решения сложных задач часто процессам нужно

взаимодействовать между собой. В связи с этим, существует необходимость в организации эффективного взаимодействия между процессами, по возможности не использующее прерываний. Основными задачами межпроцессного взаимодействия являются [14]:

- непосредственная передача информации от одного процесса другому;
- контроль над деятельностью процессов (к примеру, гарантии, что два процесса не пересекутся в критических ситуациях) для обеспечения совместной работы без создания помех друг другу;
- согласование действий процессов во времени (к примеру, процесс начинает выполнение действия, только дождавшись определенного результата другого процесса).

Данные задачи, кроме первого, относятся и к потокам.

Важнейшей проблемой межпроцессного взаимодействия является состояние состязания (гонки данных) – ситуации, где конечный результат зависит от быстродействия процессов. Данные ситуации часто происходят тогда, когда несколько процессов конкурируют между собой для получения доступа к общим ресурсам с целью их модификации. Например, несколько потоков пытаются изменить значение общей глобальной переменной. Наличие гонок данных при взаимодействии процессов приводит к нестабильности результатов вычислений при каждом новом запуске процессов. Часть кода программы, в которой есть обращение к совместно используемым данным, называется критической областью или секцией. Для предотвращения гонки данных, связанной с совместным использованием общих ресурсов, таких как память, файлов, используется взаимное исключение – запрет одновременной записи и чтения общих данных более чем одним процессом.

Данное требование предотвращает гонки данных, но только его использование недостаточно для эффективной работы параллельных процессов. Следующие условия дополняют требование взаимного исключения:

- два процесса не должны одновременно находиться в критических областях;
- в программе не должно быть предположений о производительности и

количестве процессоров в системе;

- процесс, находящийся вне критической области, не может блокировать другие процессы;
- невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.

На рисунке 15 представлена обобщенная схема реализации требования взаимоисключения. Процесс ***А*** первым попадает в критическую область в момент времени T_1. Процесс ***Б*** при попытке попасть в критическую область в момент времени $T2$ приостанавливается, до наступления момента времени T_3, когда процесс ***А*** выйдет из данной области. В момент времени T_4 процесс ***Б*** также покидает критическую область, и происходит возвращение в исходное состояние, когда ни одного процесса в критической области не было. Данные соображения справедливы и для потоков.

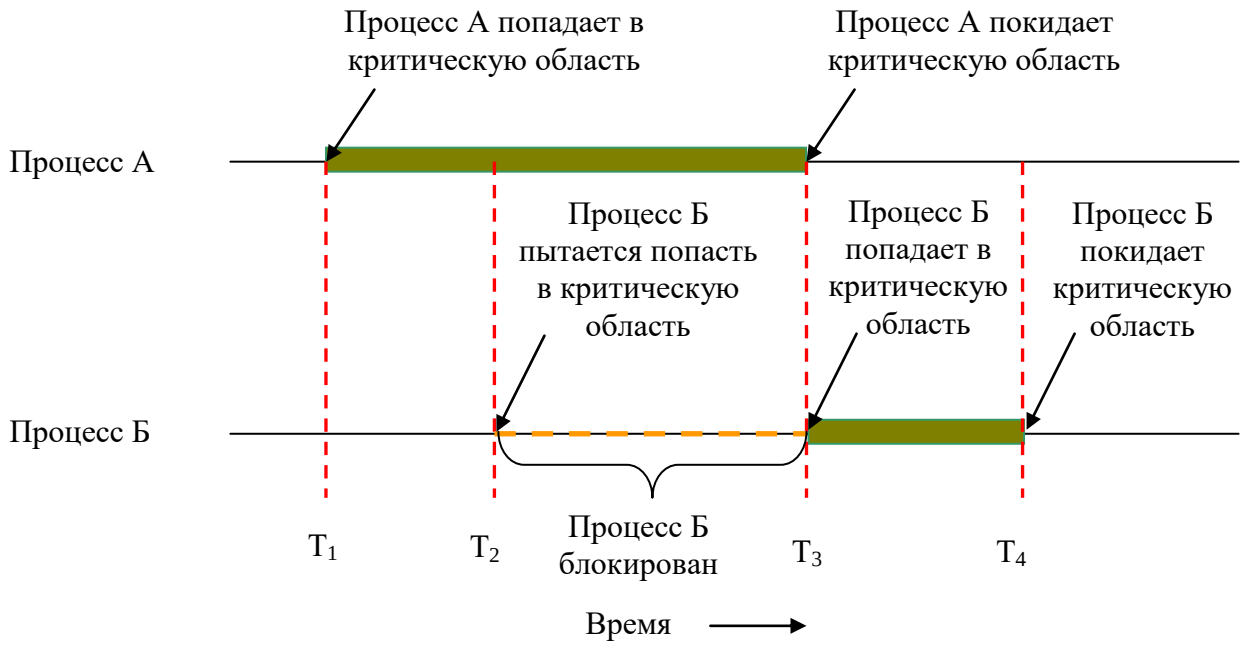


Рисунок 15 – Взаимное исключение с использованием критических областей

Для синхронизации потоков (причем, даже если потоки принадлежат разным процессам) в современных операционных системах существует следующие средства синхронизации на уровне ядра: семафоры, мьютексы, события. Данные механизмы позволяют перевести потоки в режим ожидания, без потребления времени

процессора, в случаях попытки доступа к критической области при нахождении там уже другого потока.

Семафоры.

В 1965 году Дейкстра предложил идею об использовании специальных переменных на уровне ядра ОС, доступ к которым можно получить только через атомарные операции: down (*P*) и up (*V*). При выполнении операции down значение семафора уменьшается на 1, если его значение было больше нуля, и поток продолжает работу, иначе поток блокируется. Операция up над семафором выполняет обратное действие: увеличивает значение семафора на 1. При этом если были заблокированные потоки, связанные с данным семафором, то один из них разблокируется, а его значение остается равным нулю.

Семафоры делятся на два класса: счетные и двоичные. Счетный семафор может принимать целое положительное число *N*, равное количеству элементов общего ресурса. При этом ресурс будет исчерпан только тогда, когда использованы все *N* элементов. Но часто счетных свойств семафора избыточно для большинства задач. Это те случаи, когда ресурс является неделимым. Тогда применяются мьютексы, которые могут находится только в двух состояниях: заблокированный (0) и незаблокированный (1).

События.

Представляют собой наиболее простые объекты синхронизации, использующиеся для информирования другого потока о совершении определенного события. Поток может перевести событие в «занятое» состояние и выполнить код, например, в критической области. При выходе из критической секции поток сбрасывает событие в состояние «свободно». Простейшим сценарием использования события в ОС Windows является следующий: первый поток перед входом в критическую область ожидает появления события функцией WaitForSingleObject, а другой сообщает о выполнении задания функцией SetEvent.

Таким образом, представленные средства синхронизации на уровне ядра позволяют решать задачи межпроцессного взаимодействия.

2.4 Планирование

Подсистема операционной системы, отвечающая за распределение времени процессора между процессами, называется планировщиком, а используемый алгоритм – алгоритмом планирования. Большинство процессов чередуют периоды вычислений с операциями ввода-вывода, например, чтения или записи в файл, реализуемые через системные вызовы (рисунок 16). При этом выполнение команд потока приостанавливается до завершения системного вызова ввода-вывода. Время простоя может быть значительным, так как внешние устройства обладают низкой производительностью в сравнении с ядром вычислительной системы. В связи с этим все процессы делятся на два класса:

- процессы, большую часть времени занятые вычислениями;

- процессы, в которых преобладают интервалы ожидания завершения операций ввода-вывода.

Таким образом, процессы первого класса ограниченны вычислительными возможностями процессора, вторые –возможностями устройств ввода-вывода. Данные особенности должны учитываться при планировании времени процессора.

Основные ситуации, когда необходимо принять решения по планированию процессов, является следующие:

- при создании нового процесса, необходимо определить, какой процесс запустить: родительский или дочерний;
- при завершении работы процесса необходимо из набора готовых процессов выбрать и запустить следующий, если нет ни одного готового, то запускается холостой процесс из операционной системы;
- при блокировании процесса на операции ввода-вывода, семафоре или по другой причине, необходимо выбрать и запустить другой процесс;
- при возникновении прерывания ввода-вывода, необходимо определить запускать ли процесс, который ожидал этого ввода-вывода или другой.

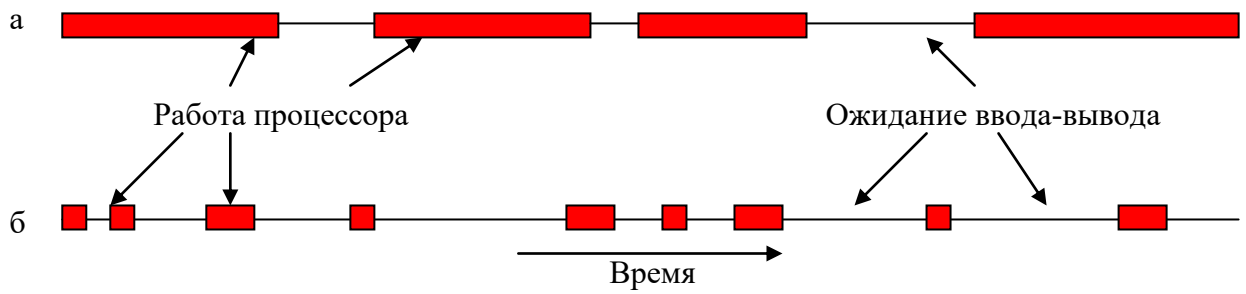


Рисунок 16 – Периоды использования процессора, чередующиеся с ожиданием ввода-вывода: процесс, ограниченный возможностями процессора (а); процесс, ограниченный возможностями устройств ввода-вывода (б)

Системы планирования современных ОС базируются на работе аппаратного таймера, который отсчитывает заданный интервал времени и генерирует периодические прерывания с определенной частотой. Планировщик принимает решения по выбору процесса для предоставления ему времени процессора при каждом прерывании от таймера. При этом алгоритмы планирования делятся на два основных класса согласно их реакции на прерывания об истечении заданного кванта времени от таймера:

- алгоритмы планирования без переключений (неприоритетное планирование). Выбранный операционной системой процесс работает либо до блокировки (в ожидании ввода-вывода или другого процесса), либо до тех пор, пока процесс сам не отдаст процессор. Данный вид планировщика используется в операционных системах с невытесняющей многозадачностью. Недостатком данного подхода является то, что ошибки в коде программы могут привести к бесконечным циклам, что приведет к блокировке всего компьютера.

- алгоритмы планирования с переключениями (приоритетное планирование). Выбранный процесс работает заданный интервал времени, по истечении которого планировщик приостанавливает его работу и передает управление другому. Приоритетное планирование использует прерывания от таймера, чтобы передать управление другому процессу. При использовании данного подхода сбой в работе приложения не приведет к тому, что другие процессы не получат времени процессора.

Различные алгоритмы планирования предназначены и эффективны для определенных систем. Выделяют три основных класса вычислительных систем [20]:

- системы пакетной обработки данных;
- интерактивные системы;
- системы реального времени.

В системах первого класса нет пользователей за терминалами, поэтому в реализации многозадачности для обеспечения их удобства работы нет необходимости. Основными критерием эффективности подобных систем является минимизации времени решения всех задач. Поэтому приемлемы алгоритмы без переключений или с переключениями, но с большим временем процессора, отводимым каждому процессу.

В интерактивных системах необходимы алгоритмы с переключениями, чтобы предотвратить захват процессора одним процессом.

В системах реального времени приоритетное планирование необязательно, так как там существуют другие программы, которые знают когда надо блокироваться.

Критерии эффективности системы планирования могут быть общими, так и специфическими для конкретного класса систем. Ниже представлены основные критерии эффективности планировщика:

а) для всех систем;

1) справедливость – предоставление каждому процессу справедливой доли процессорного времени;

2) принудительное применение политики – контроль за выполнением принятой политики (к примеру, предоставление процессам контроля безопасности процессора по первому требованию);

3) баланс – поддержка занятости всех частей системы (важно, чтобы работало больше устройств, чем один процесс только производил вычисления);

б) для систем пакетной обработки данных;

1) пропускная способность – максимальное количество задач в час;

2) оборотное время – минимизация времени, затрачиваемого на ожидание обслуживания и обработку задачи;

3) использование процессора – поддержка постоянной занятости процессора;

в) для интерактивных систем;

1) время отклика – быстрая реакция на запросы;

2) соразмерность – выполнение пожеланий пользователя;

г) для систем реального времени;

1) окончание работы к сроку – предотвращение потери данных;

2) предсказуемость – предотвращение деградации качества в мультимедийных системах.

Ниже рассмотрены популярные алгоритмы планирования процессов, которые справедливы и для потоков.

2.4.1 Планирование в системах пакетной обработки данных

«Первым пришёл – первым ушёл»(First Input –First Output, FIFO).

Представляет собой простой алгоритм планирования с точки зрения реализации. Данный алгоритм планирования относится к классу без переключений. Процессы получают время процессора в том порядке, в котором они его запросили. Формируется единая очередь процессов. Когда текущий процесс блокируется, то запускается следующий в очереди, а когда блокировка снимается, процесс попадает в конец очереди. Основным недостатком данного алгоритма является то, что если в очереди присутствуют процессы, ограниченные устройствами ввода-вывода (т.е. большую часть времени тратящие на ожидание завершения команд ввода-вывода) и производительностью процессора, то это приводит к увеличению времени выполнения всего пакета заданий. Это связано с тем, что процессы, ограниченные производительностью процессора, долго работают и не позволяют поставить в очередь запросы на выполнение медленных команд ввода-вывода другого класса процессов. Было бы более эффективным прерывать долгие вычисления процесса,

чтобы предоставить другому процессу, ограниченному устройством ввода-вывода, возможность инициировать системный вызов.

2 Алгоритм «Кратчайшая задача – первая»(Shortest Job First).

Данный алгоритм также относится к типу без переключений. Основная идея алгоритма заключается в следующем: если в очереди есть несколько важных задач, то планировщик выбирает первой самую короткую по ожидаемому времени выполнения. Эта схема работает лишь в случае одновременного наличия нескольких задач и обычно неактуальна.

3 Алгоритм «Наименьшее оставшееся время выполнения»

Является разновидностью предыдущего алгоритма. Планировщик каждый раз выбирает процесс с наименьшим оставшимся временем выполнения. Для данного вида алгоритма необходимо знать, сколько времени выполняются процессы и сколько еще осталось, что обычно является сложной задачей.

4 Алгоритм трехуровневого планирования

Системы пакетной обработки позволяют реализовать систему планирования, разделенную на 3 уровня, как представлено на рисунке 17. Новые задачи сначала помещаются в очередь, хранящуюся на диске. Планировщик заданий, с учетом заданного приоритета – по времени выполнения или как-то по другому (например, по работе с устройствами ввода-вывода), выбирает задачу и отправляет на исполнение.

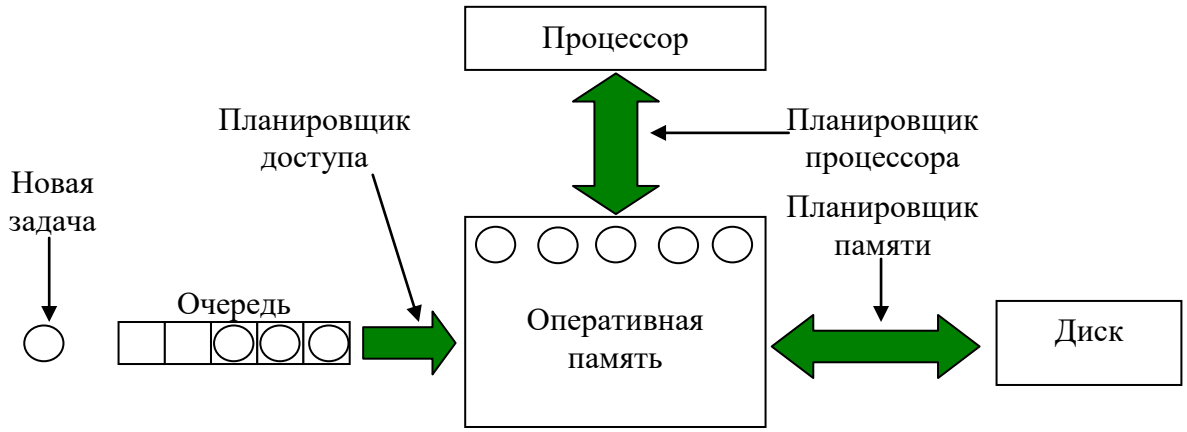


Рисунок 17 – Трехуровневое планирование

Созданный процесс для выбранной задачи вступает в борьбу за доступ к процессору. На втором уровне планирования (планировщик памяти) при нехватке памяти определяются, какие процессы будут находиться в ОЗУ, а какие можно выгрузить на диск. Данные перемещения не должны происходить слишком часто, так как дисковые операции являются сравнительно медленными. Степень многозадачности определяется как количество процессов, одновременно находящихся в оперативной памяти.

На третьем уровне планирования выбирается процесс, готовый к исполнению, для доступа к процессору. При этом планировщик может использовать наиболее подходящий к ситуации алгоритм планирования, как с прерыванием, так и без.

2.4.2 Планирование в интерактивных системах

В интерактивных системах невозможно реализовать долгосрочное планирование задач, поэтому применяется двухуровневая схема (планировщик памяти и процессора). Ниже рассмотрены наиболее популярные алгоритмы планирования. Причем алгоритмы планирования для интерактивных систем могут использоваться и в системах пакетной обработки.

Алгоритм циклического планирования.

Каждому процессу предоставляется заданный квант времени процессора. Если к концу кванта времени процесс всё ещё работает, то он прерывается и отправляется в конец списка, а управление передается следующему. При этом в очереди находятся только процессы с состоянием «Готов». Вновь созданные или перешедшие в готовое состояние процессы становятся в конец очереди. На рисунке 18 представлена иллюстрация циклического планирования.

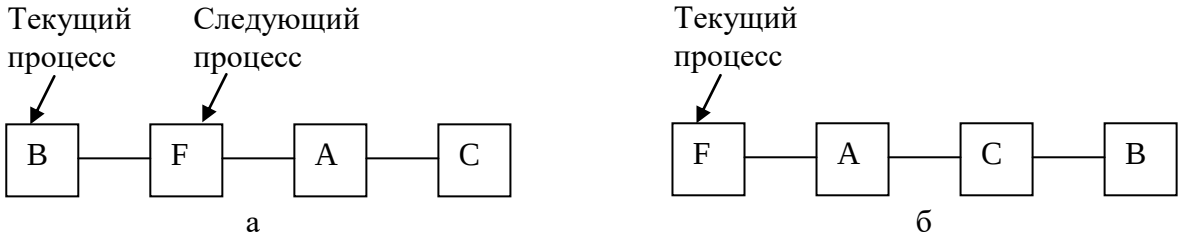


Рисунок 18 – Циклическое планирование

Выбор длительности кванта времени влияет на эффективность алгоритма планирования: слишком малый квант приведёт к частому переключению процессов и увеличению служебных расходов, а слишком большой - может привести к медленному реагированию на короткие запросы интерактивных приложений.

Алгоритм с приоритетным планированием.

Циклический алгоритм предполагает, что все процессы имеют одинаковое значение для компьютера. На самом деле в системе присутствуют процессы, которые имеют приоритет перед другими, например, системные службы ОС перед пользовательскими приложениями. Основная идея алгоритма – управление передается процессу с самым высоким приоритетом. Фактически, в системе для каждого уровня приоритета присутствует своя очередь. В рамках каждой очереди выбор следующего процесса производится на основе циклического планирования (рисунок 19). При наличии процессов с состоянием «Готов» из разных очередей будет выбираться процесс с более высоким приоритетом.

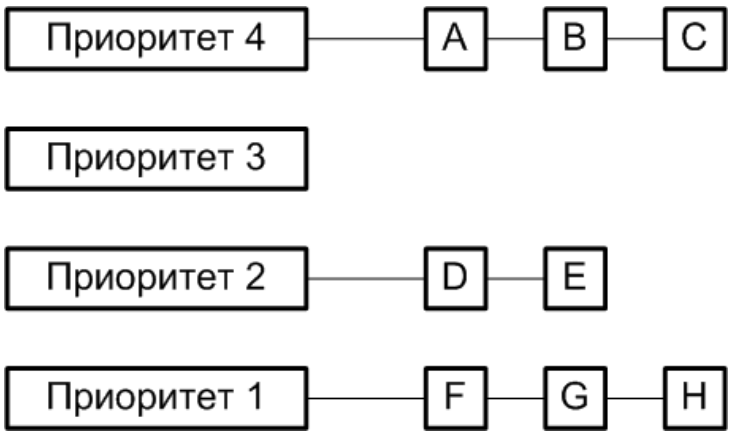


Рисунок 19 – Схема приоритетного планирования

Основным недостатком данного алгоритма планирования является то, что если в системе преобладают высокоприоритетные процессы, то это приведет к ситуации, когда низкоприоритетные не будут получать время процессора. Данная ситуация называется «голоданием» процесса. Чтобы предотвратить подобную ситуацию, часто используются схемы с адаптивным или динамическим изменением приоритета, учитывающие, в частности, время работы процесса. Например, если

процесс имел низкий уровень приоритета и долго не получал время процессора, то его уровень будет ОС повышаться. При получении достаточного времени процессора уровень приоритета процесса снова снижается до базового.

Алгоритм «Самый короткий процесс – следующий».

Похож на алгоритм планирования «Кратчайшая задача – первая». Предполагается запускать на выполнение тот процесс, который потребует минимальное время выполнения. Но, в отличие от систем с пакетной обработкой, в интерактивных системах не известно время выполнения задачи. В связи с этим, для определения времени исполнения процесса в таких системах используется метод оценки по предыдущим запускам программы, который получил название метод оценки с учетом старения.

Допустим, что время исполнения первого запроса составило значение $\boldsymbol{T_0}$, а второго $\boldsymbol{T_1}$. Тогда оценку времени выполнения третьего запроса $\boldsymbol{T_3}$ можно рассчитать, взяв взвешенную сумму этих интервалов: $\alpha\boldsymbol{T_0} + (1 - \alpha)\ \boldsymbol{T_1}$. Параметр α позволяет либо заставить быстро забывать о предыдущих запусках или, наоборот, помнить о них в течение долгого времени. Взяв α = 1/2, получим серию оценок:

$$T_0, \frac{T_0}{2} + \frac{T_1}{2}, \frac{T_0}{4} + \frac{T_1}{4} + \frac{T_2}{2}, \frac{T_0}{8} + \frac{T_1}{8} + \frac{T_2}{4} + \frac{T_3}{2}. \quad (1)$$

Как видно из выражения (1), в каждом следующем запуске степень влияния предыдущих оценок снижается.

Гарантированное планирование.

Идеей данного алгоритма заключается в том, чтобы гарантировать всем процессам приблизительно 1/**N** часть времени процессора, где $\boldsymbol{N}$- количество запущенных процессов. Чтобы выполнить это требование, система должна отслеживать распределение времени процессора между процессами с момента создания каждого процесса. Введем следующие параметры процесса: время нахождения процесса в ОС $\boldsymbol{T_i}$ и суммарное значение выделенного времени процессора $\boldsymbol{t_i}$. Основное требование данного алгоритма определяется следующим выражением:

$$t_i \approx T_i/N. \quad (2)$$

Для выполнения требования (2) планировщик выбирает тот процесс на выполнение, у кого коэффициент справедливости принимает наименьшее значение:

$$\frac{t_i \cdot N}{T_i} \rightarrow min. \quad (3)$$

Лотерейное планирование.

В основе данного алгоритма лежит принцип случайного доступа к различным ресурсам, в том числе и к процессору. При выборе процесса планировщик выбирает случайный лотерейный билет, обладатель которого получает доступ к процессору. Розыгрыш «билетов» производится периодически, например 100 раз в секунду. Тогда победитель каждого розыгрыша получит 10 мс времени процессора. При условии, что общее число билетов равно 100 и 15 из них принадлежат одному процессу, то ему достанется приблизительно 15 % времени процессора. Таким образом, уровень приоритета процессов определяется количеством билетов.

Справедливое планирование.

Некоторые системы обращают внимание на владельца процесса перед планированием. В такой модели каждому пользователю достается некоторая доля процессора, и планировщик выбирает процесс в соответствии с этим фактом. Если в нашем примере каждому из пользователей было обещано по 50 % времени процессора, то данная доля времени им достанется независимо от количества запущенных процессов.

2.4.3 Планирование в системах реального времени

Системы реального времени (СРВ) ограничены по времени выполнения определенного запроса. Выделяют жесткие системы реального времени, в которых недопустимо выход за пределы интервала обработки, и гибкие системы реального времени, в которых нарушение графика работы нежелательно. Соблюдение графика определяется работой планировщика операционной системы.

События, на которые система реального времени должна реагировать, делятся на два типа: периодические и непериодические. В зависимости от количества

обрабатываемых событий и затрачиваемого времени система может оказаться перегруженной и не в состоянии обеспечить режим реального времени. Условие, по которому СРВ считается поддающейся планированию, является следующее:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1. \qquad (2)$$

где m- количество периодических событий;

P_i – период ***i***-го события;

C_i – время обработки ***i***-го события.

Все потоки могут быть своевременно обработаны только при выполнении данного условия.

2.5 Понятие взаимоблокировки

При взаимодействии двух или более процессов возможны патовые ситуации, заключающиеся в следующем: процесс, заняв ресурс ***A***, пытается получить доступ к ресурсу ***B***, а другой процесс наоборот – занял ресурс ***B*** и пытается получить доступ к ***A***. Это приводит к ситуации блокировки обоих процессов, из которой они не смогут выйти самостоятельно. Такая ситуация называется тупиковой или взаимоблокировкой.

Например, первый процесс запрашивает сканер и получает его. В это время второй процесс запрашивает доступ DVD-диску и получает его. Затем оба пытаются получить доступ к DVD и сканеру и блокируются, ожидая пока устройства освободятся. Тупиковая ситуация может возникнуть тогда, когда процессам предоставляются исключительные права доступа ресурсам компьютера: устройствам, файлам и т.д. Ресурсы бывают двух основных типов:

- выгружаемые (оперативная память). Данный ресурс может безболезненно быть отобран у процесса. Примером подобного ресурса является оперативная память персональных компьютеров. При истечении выделенного кванта времени процесса его данные из ОЗУ могут быть выгружены на жесткий диск. При

переключении на данный процесс его данные обратно копируются в оперативную память и его работа продолжается;

- невыгружаемые (DVD-дисковод). Данный вид ресурса нельзя отобрать у процесса, так как это может привести к сбою его работы. Например, если отнять у процесса, осуществляющего запись информации на DVD-диск, привод, то это может привести к потере данных и порче диска.

Проблемы взаимоблокировки характерны для невыгружаемых ресурсов.

Определение: группа процессов находится в тупиковой ситуации, если каждый процесс из группы ожидает события, которое может вызвать только другой процесс из той же группы.

Для возникновения ситуации взаимоблокировки должны выполняться следующие условия [22]:

1 *Условие взаимного исключения.* Каждый ресурс в данный момент или отдан ровно одному процессу, или доступен.

2 *Условие удержания и ожидания.* Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.

3 *Условие отсутствия принудительной выгрузки ресурса*. У процесса нельзя принудительным образом забрать ранее полученные ресурсы. Процесс, владеющий ими, должен сам освободить их.

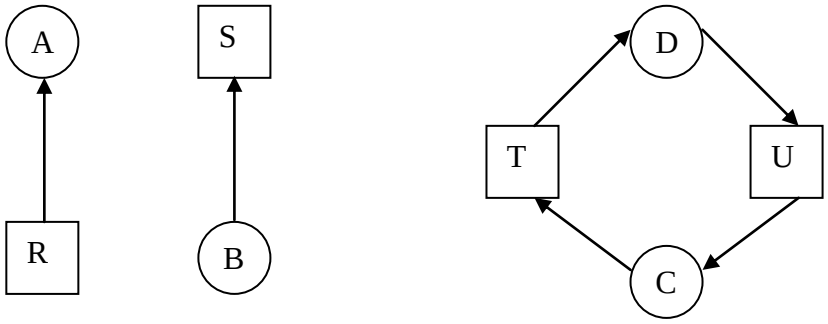
4 *Условия циклического ожидания.* Должна существовать круговая последовательность из двух и более процессов, каждый из которых ждёт доступа к ресурсу, удерживаемому следующим членом последовательности.

Таким образом, для исключения тупиковой ситуации достаточно исключить хотя бы одно из этих условий.

Для моделирования взаимоблокировок используются графы с двумя видами узлов: процессы (кружочки) и ресурсы (квадратики). Ребро, направленное от узла ресурса (квадрат) к узлу процесса (круг), означает, что ресурс ранее был запрошен и и получен в данный момент этим процессом. На рисунке 20 приведены возможные варианты графов.

На рисунке 21 изображена ситуация возникновения взаимоблокировки.

Каждый процесс получил по одному ресурсу (ребра 1-3) процесс получает свой ресурс. На шагах 4-6 каждый процесс впадает в режим ожидания освобождения ресурс, что приводит к взаимоблокировке. Данной ситуации можно избежать, если на время заблокировать один из процессов. Таким образом, если в графе есть цикл, то возможно существует тупиковая ситуация.



а б в

Рисунок 20 – Графы распределения ресурсов: ресурс R отдан процессу A (а); процесс B ждёт ресурса S (б); взаимоблокировка (в)

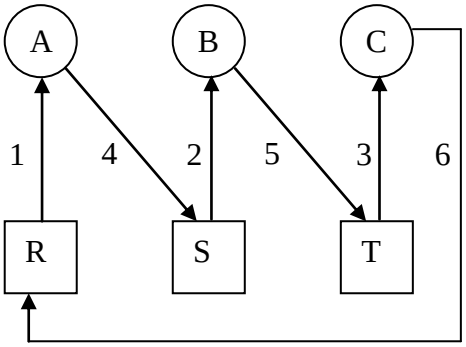


Рисунок 21 – Ситуация возникновения взаимоблокировки

Существуют 4 стратегии действий при взаимоблокировке:

1 Пренебрежение проблемой в целом. Если вы проигнорируете проблему, то возможно в дальнейшем она проигнорирует вас.

2 Обнаружение и восстановление. Позволить взаимоблокировке произойти, обнаружить её и предпринять какие-либо действия.

3 Динамическое избегание тупиковых ситуаций с помощью аккуратного распределения ресурсов.

4 Предотвращение с помощью структурного опровержения одного из

четырёх условий, необходимых для взаимоблокировки.

Большая часть операционных систем, включая UNIX и Windows, игнорируют проблему взаимоблокировки, так как исходят из предположения, что большинство пользователей предпочтут иметь дело со случающимися время от времени взаимоблокировками, чем руководствоваться правилом, ограничивающим использование ресурсов.

2.6. Лабораторная работа «Управление процессами в Windows»

2.6.1 Теоретическая часть

В операционной системе Windows реализован механизм вытесняющего планирования на основе приоритетов, позволяющий выбирать поток с наибольшим приоритетом, готовый к выполнению. Выбранный для выполнения поток работает в течение заданного кванта времени. По окончании кванта ОС выбирает готовый к выполнению другой поток с таким же (или большим) уровнем приоритета. При отсутствии подобных потоков текущему выделяется еще один интервал времени. Текущий поток может быть досрочно прерван, когда поток с более высоким приоритетом перейдет в состояние готов к выполнению.

Основной API-функцией для создания процесса в Windows является **CreateProcess**. Ниже представлен интерфейс данной функции:

```
BOOL CreateProcess
(
LPCTSTR lpApplicationName,                    // имя исполняемого модуля
LPTSTR lpCommandLine,                         // Командная строка
LPSECURITY_ATTRIBUTES lpProcessAttributes,    // Указатель на структуру
                                              // SECURITY_ATTRIBUTES
LPSECURITY_ATTRIBUTES lpThreadAttributes,     // Указатель на структуру
                                              // SECURITY_ATTRIBUTES
```

```
BOOL bInheritHandles,                    // Флаг наследования текущего
процесса
DWORD dwCreationFlags,                   // Флаги способов создания
процесса
LPVOID lpEnvironment,                    // Указатель на блок среды
LPCTSTR lpCurrentDirectory,              // Текущий диск или каталог
LPSTARTUPINFO lpStartupInfo,             // Указатель нас структуру
STARTUPINFO
LPPROCESS_INFORMATION lpProcessInformation   // Указатель нас структуру
                                         // PROCESS_INFORMATION
);
```

Подробное описание параметров данной функции представлено в MSDN. При успешном запуске процесса данная функция возвращает значение TRUE. Ниже представлен фрагмент кода, иллюстрирующий пример создания и уничтожения процесса программы Notepad:

```
STARTUPINFO cif;
ZeroMemory(&cif,sizeof(STARTUPINFO));
PROCESS_INFORMATION pi;
if (CreateProcess("c:\\windows\\notepad.exe",NULL,
        NULL,NULL,FALSE,NULL,NULL,NULL,&cif,&pi)==TRUE)
{
        cout << "process" << endl;
        cout << "handle " << pi.hProcess << endl;
        Sleep(3000);                     // подождать
        TerminateProcess(pi.hProcess,NO_ERROR);      // убрать процесс
}
```

Листинг 1. - Пример использования функции CreateProcess

В данном примере запускается на выполнение стандартная программа Windows «Блокнот» и через 3 секунды данный процесс уничтожается. Для уничтожения процесса используется функция **TerminateProcess**, которая в качестве параметра принимает дескриптор процесса. Ниже приведен синтаксис функции **TerminateProcess**:

```
BOOL TerminateProcess
(
    HANDLE hProcess,       // Указатель процесса
    UINT uExitCode  // Код возврата процесса
);
```

При успешном выполнении функция возвращает истину.

При необходимости удалить процесс, который не был создан в текущем приложении, возникает задача определить его дескриптор. Для этого можно использовать следующий алгоритм:

- при помощи функции **FindWindowEx**, передавая ей заголовок окна процесса, найти идентификатор окна программы;
- определить идентификатор потока, создавшего окно программы (GetWindowThreadProcessId);
- получить открытый дескриптор заданного процесса (**OpenProcess**).

Для получения информации о временных параметра процесса можно воспользоваться следующей функцией:

```
BOOL GetProcessTimes
(
    HANDLE hProcess,                // дескриптор процесса
    LPFILETIME lpCreationTime,      // время создания процесса
    LPFILETIME lpExitTime,          // время выхода из работы процесса
    LPFILETIME lpKernelTime,   // время, работы процесса в режиме ядра
    LPFILETIME lpUserTime           // время, работы процесса в режиме
пользователя
);
```

Таким образом, представленные функции позволяют управлять процессами и определять их параметры.

2.6.2 Постановка задачи

Написать консольное приложение, позволяющее:

- определить время создания внешнего процесса, время работы в режимах яра и пользовательском;

- завершить выполнение внешнего процесса.

В качестве внешних процессов можно использовать следующие заранее запущенные стандартные программы Windows: калькулятор, paint, блокнот, Word, Excel.

2.6.3 Ход выполнения работы

1. Создайте в Visual Studio проект из шаблона «Visual C++| Консольное приложение».

2. Разработайте код, позволяющий решить поставленные задачи. Возможный вариант кода представлен ниже:

```
#include "stdafx.h"
#include <Windows.h>
#include <iostream>
int main(int argc, char* argv[])
{
	//HWND hwnd = FindWindow(NULL, "Калькулятор");
	HWND hwnd = FindWindow(NULL, "Безымянный - Paint");
	if (!hwnd)
	{
		std::cout << "Error find window\n";
		return 1;
	}
	std::cout << "HWND=" << std::hex << hwnd << "\n";
	DWORD id;
	GetWindowThreadProcessId(hwnd, &id);
	HANDLE hProcess;
	hProcess=OpenProcess(PROCESS_ALL_ACCESS, false, id);
	if (!hProcess)
```

```
	{
		std::cout << "Error get handle process\n";
		return 1;
	}
	std::cout << "hProcess=" << std::hex << hProcess << "\n";
	FILETIME ft[4];
	GetProcessTimes(hProcess, &ft[0], &ft[1], &ft[2], &ft[3]);
	SYSTEMTIME tm[4];
	FileTimeToLocalFileTime(&(ft[0]), &(ft[0]));
	for (int i = 0; i < 4; i++)
		FileTimeToSystemTime(&ft[i], &tm[i]);
	printf("Created at %02d:%02d:%02d\n", tm[0].wHour, tm[0].wMinute,
tm[0].wSecond);
	printf("Kernel time %02d:%02d:%02d:%02d\n", tm[2].wHour, tm[2].wMinute,
tm[2].wSecond, tm[2].wMilliseconds);
	printf("User time %02d:%02d:%02d:%02d\n", tm[3].wHour, tm[3].wMinute,
tm[3].wSecond, tm[3].wMilliseconds);
	TerminateProcess(hProcess, NO_ERROR);
	CloseHandle(hProcess);
	return 0;
```

3. Запустите внешнюю программу, например, Paint.
4. Запустите разработанную программу и убедитесь в её работоспособности:

   - выводится информация о внешнем процессе;

   - внешний процесс уничтожается.
5. Сделайте выводы по работе и оформите отчет.

2.7 Контрольные вопросы

1. Поясните термины «программа», «процесс» и «поток».
2. Стек является общим для всех потоков?
3. Поясните термин псевдопараллелизм.
4. Что такое квант времени?
5. Что такое аппаратный контекст потока?
6. В чем суть механизма переключения процесса?
7. В чем преимущества многопоточного приложения перед множеством взаимодействующих процессов?
8. Как используются прерывания от таймера для реализации вытесняющей

многозадачности?

9. Что такое состояние состязания?

10. Дайте определение критической секции.

11. Определите понятие семафор?

12. Чем отличается семафор от мьютекса?

13. Представьте себе операционную систему, разработанную для компьютера, в котором отсутствует система прерываний. Какой алгоритм планирования процессов может быть реализован в такой системе?

14. Приведите пример алгоритма планирования, в результате работы которого процесс, располагая всеми необходимыми ресурсами, может бесконечно долго находиться в системе, не имея возможности завершиться.

15. Пусть в системе существует два процесса и три одинаковых ресурса. Каждому процессу требуется два ресурса. Возможна ли взаимоблокировка? Объясните ваш ответ.

3 Управление памятью

3.1 Основы управления памятью

Основная память компьютера делится на две части: участки операционной системы и пользовательских процессов. В многозадачных ОС необходимо распределить множество процессов в пользовательской части памяти. Данной задачей занимается подсистема управления памятью (менеджер памяти), основными функциями которой в мультипрограммной системе являются:

- отслеживание свободных и занятых участков памяти;
- выделение памяти процессам и освобождение памяти по завершении работы процессов;
- вытеснение долго не использующихся процессов из ОЗУ на диск (полное или частичное), когда размеры основной памяти не достаточны для

размещения в ней всех процессов, и возвращение их обратно, когда в памяти освобождается место;

- настройка адресов программы на конкретную область физической памяти [16].

Для адресации переменных и команд на разных этапах жизненного цикла программы используются символьные имена (метки), виртуальные адреса и физические адреса (рисунок 22).

Символьные имена используются пользователем при написании программ на языках высокого уровня и ассемблере. Использование символьных обозначений переменных, функций упрощает восприятие кода программистом. Транслятор, преобразующий программный код в машинный формат, переводит символьные идентификаторы в виртуальные адреса, иногда называемые математическими, или логическими адресами. Во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа. Поэтому транслятор присваивает переменным и командам виртуальные (условные) адреса, представляющие собой смещение от начала диапазона памяти, выделенного под процесс. При выполнении же команд процессор использует физические адреса, которые формируются на основе виртуальных адресов и данных о начале участка памяти процесса.

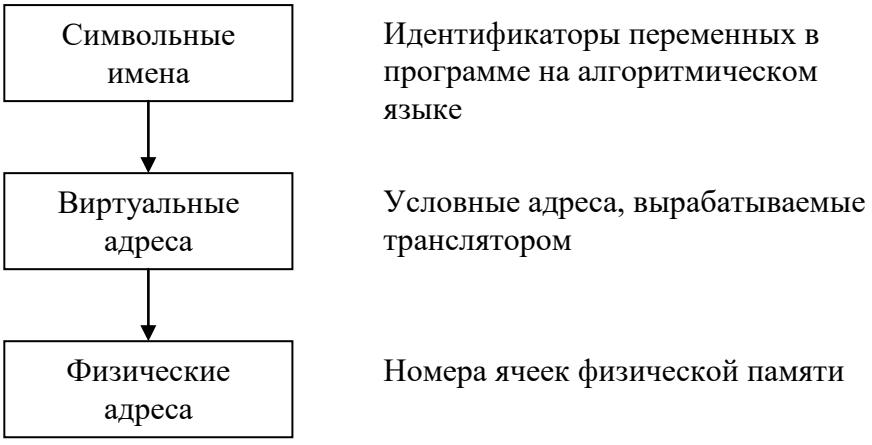


Рисунок 22 – Типы адресов

Диапазон виртуальных адресов процесса называется виртуальным адресным пространством. Фактически, виртуальное адресное пространство у всех процессов является одним и тем же. Но возможное совпадение виртуальных адресов переменных и команд различных процессов не приводит к конфликтам, так как ОС отображает их на разные физические адреса.

Существуют два основных подхода к преобразованию виртуальных адресов в физические. В первом случае осуществляется замена виртуальных адресов на физические во время загрузки программного кода в память. В дальнейшем будут использоваться только физические адреса.

Во втором случае, программа загружается в память с неизменными виртуальными адресами операндов инструкций и переходов, значения которых были выработаны транслятором. В простейшем случае, когда виртуальная и физическая память процесса представляют собой непрерывные области адресов, ОС выполняет преобразование виртуальных адресов в физические так, как представлено на рисунке 23. При загрузке программы операционная система фиксирует адрес начала сегмента памяти процесса ***S***. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. При этом виртуальный адрес *VA* рассматривается как смещение относительно физического адреса начала блока памяти процесса.

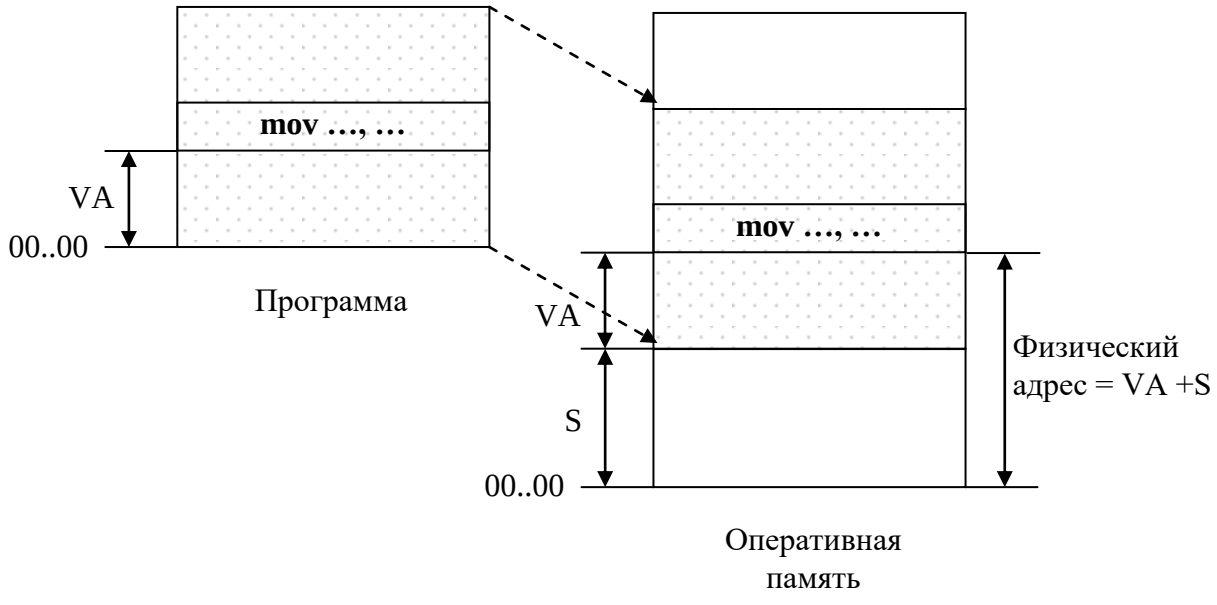


Рисунок 23 – Схема динамического преобразования адресов

Простейшая система распределения памяти представляет собой однозадачную систему без подкачки на диск. В таком случае в каждый момент времени работает только один процесс. Простейшие способы организации памяти для подобных систем, представлены на рисунке 24.

Рисунок 24 –Модели организации памяти для однозадачной системы

Первая модель использовалась на старых компьютерах, вторая - сейчас на некоторых встроенных системах. Третья модель устанавливалась на ранних персональных компьютерах, оснащенных MS-DOS, где в роли ПЗУ выступает BIOS.

Современные вычислительные системы позволяют запускать одновременно несколько программ, что определяет необходимость использования более сложных алгоритмов управления памятью. Менеджеры памяти делятся на два класса по методам распределения памяти:

- с использованием внешней памяти;
- без использования внешней памяти (рисунок 25).

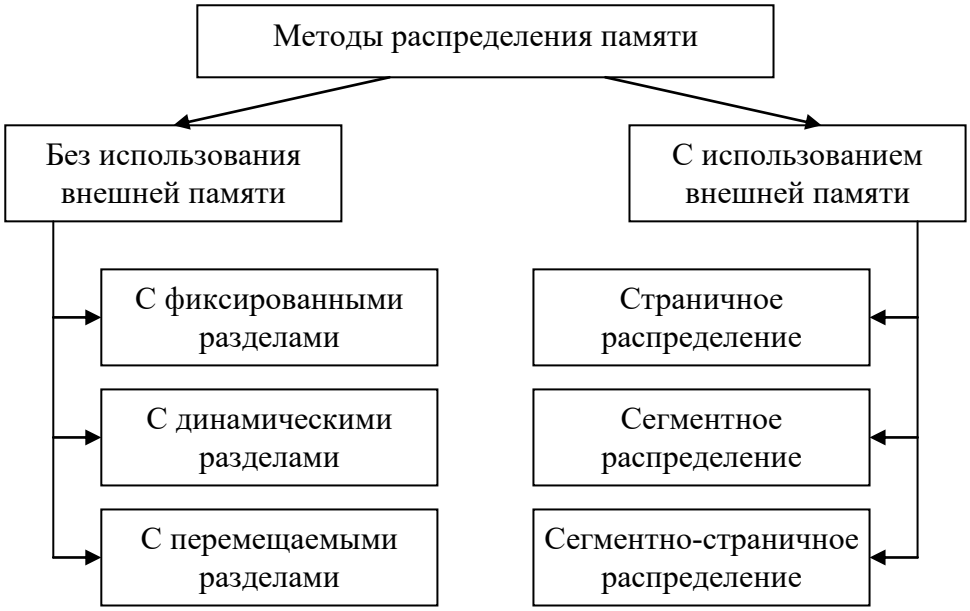


Рисунок 25 – Методы распределения памяти

Представленные классы систем управления памятью будут рассмотрены более подробно в следующих разделах.

3.2 Методы распределения памяти без использования подкачки

3.2.1 Метод распределения с фиксированными разделами

Первой многозадачной системой была система с фиксированными разделами. Память компьютера разбивается на несколько участков с фиксированными размерами, в дальнейшем называемыми разделами. Конфигурирование разделов может быть выполнено вручную пользователем во время старта компьютера или во время установки операционной системы. В дальнейшем размеры разделов остаются постоянными.

Вновь созданный процесс помещается в очередь доступа к памяти: либо в общую (рисунок 24, а), либо к определенному разделу (рисунок 24, б).

Менеджер памяти в этом случае выполняет следующие операции:

- ищет свободный раздел памяти размером, достаточным для загрузки нового процесса;

- осуществляет загрузку программы в свободный раздел и настройку адресов.

У программиста существует возможность определить для программы конкретный раздел памяти для её исполнения. Это позволяет получить машинный код, уже настроенный на конкретную область памяти и не требующий преобразования виртуальных адресов в физические.

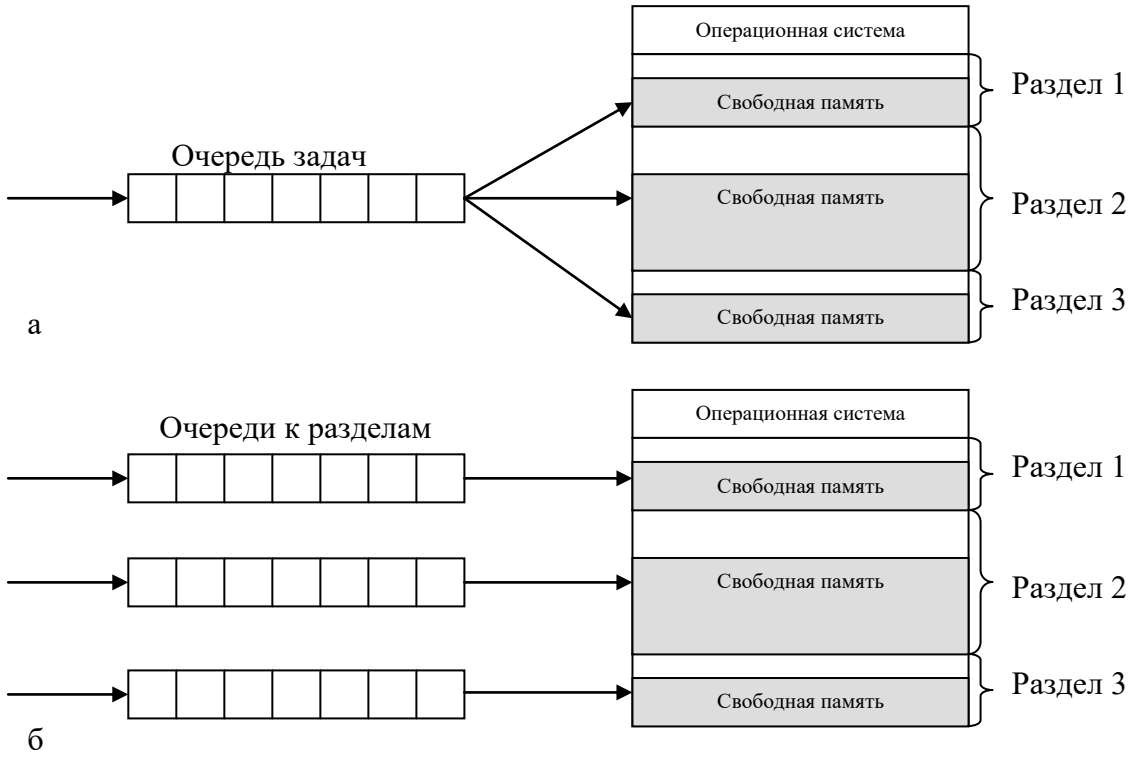


Рисунок 24 – Распределение памяти фиксированными разделами: с общей очередью (а), с отдельными очередями (б)

Основным преимуществом данного метода распределения памяти является простота реализации. Но применение фиксированных разделов приводит к неэффективному использованию памяти в тех случаях, когда процессам требуется меньший объем памяти, чем величина раздела. Также привязка программ к конкретным разделам может привести к длительным ожиданиям при использовании одного раздела несколькими процессами. Данная схема была применена в OS/360. В настоящий момент практически нигде не используется.

3.2.2 Метод распределения с динамическими разделами

В этом случае формирование разделов осуществляется во время работы по мере необходимости. Первоначально участок памяти вычислительного устройства, выделенный для пользовательских процессов, определяется свободным. Для каждого нового процесса операционная система выделяет раздел с необходимым объемом памяти (если отсутствует достаточный свободный объем памяти, то программа не запускается). Необходимо отметить, что размеры разделов могут быть разными. После завершения процесса память освобождается и на это место может быть загружен другой процесс.

На рисунке 25 представлена схема работы подсистемы управления памятью при использовании динамического распределения. В момент времени t_0 в памяти находится только ОС. К моменту t_1 в память загружены 5 процессов. В следующий момент процесс ***П4*** завершается и выгружается из памяти. На освободившееся от процесса ***П4*** место загружается процесс ***П6***, поступивший в момент t_3.

Основными задачами операционной системы, реализующей данный метод управления памятью, являются следующие:

- формирование таблиц с информацией о свободных и занятых областях с указанием адреса начала раздела и его размера;
- при запуске процесса — поиск свободного участка памяти, размер которого достаточен для размещения нового процесса. Могут использоваться различные алгоритмы выбора раздела: «первый попавшийся раздел достаточного размера», «раздел, имеющий наименьший достаточный размер» или «раздел, имеющий наибольший достаточный размер»;
- загрузка программы в выделенный ей раздел памяти и корректировка таблиц свободных и занятых областей. Данная модель базируется на принципе, что процессы не перемещаются в другие разделы памяти во время выполнения. Данное обстоятельство позволяет осуществить преобразование адресов однократно;
- после выгрузки процесса из памяти осуществляется корректировка таблиц свободных и занятых областей.

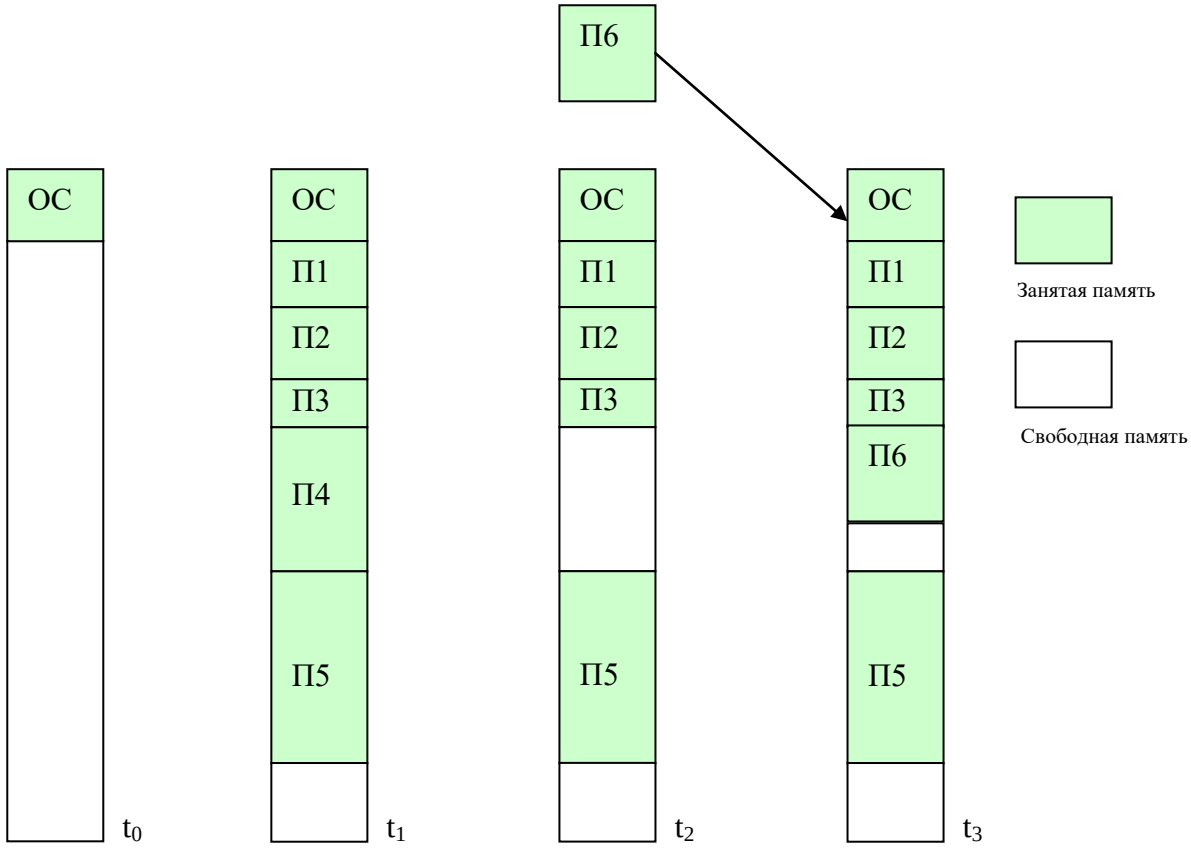


Рисунок 25 – Распределение памяти динамическими разделами

По сравнению с методом распределения памяти фиксированными разделами данный подход обладает большей гибкостью по распределению памяти. Основной недостаток – фрагментация памяти, заключающаяся в возможности появления большого числа несмежных участков свободной памяти маленького размера (фрагментов). Это приводит к тому, что суммарный объем свободной памяти достаточно большой, но загрузить процесс не получается.

3.2.3 Метод распределения с перемещаемыми разделами

Для предотвращения фрагментации памяти может быть использован следующий подход: перемещение всех занятых разделов в сторону старших или младших адресов, чтобы вся свободная память образовала единую область (рисунок 26). Таким образом, к стандартным функциям ОС при распределении памяти динамическими разделами добавляется задача периодической дефрагментации

памяти за счет копирования разделов из одного места в другое, корректируя при этом таблицы свободных и занятых областей. Данная процедура получила название сжатие.

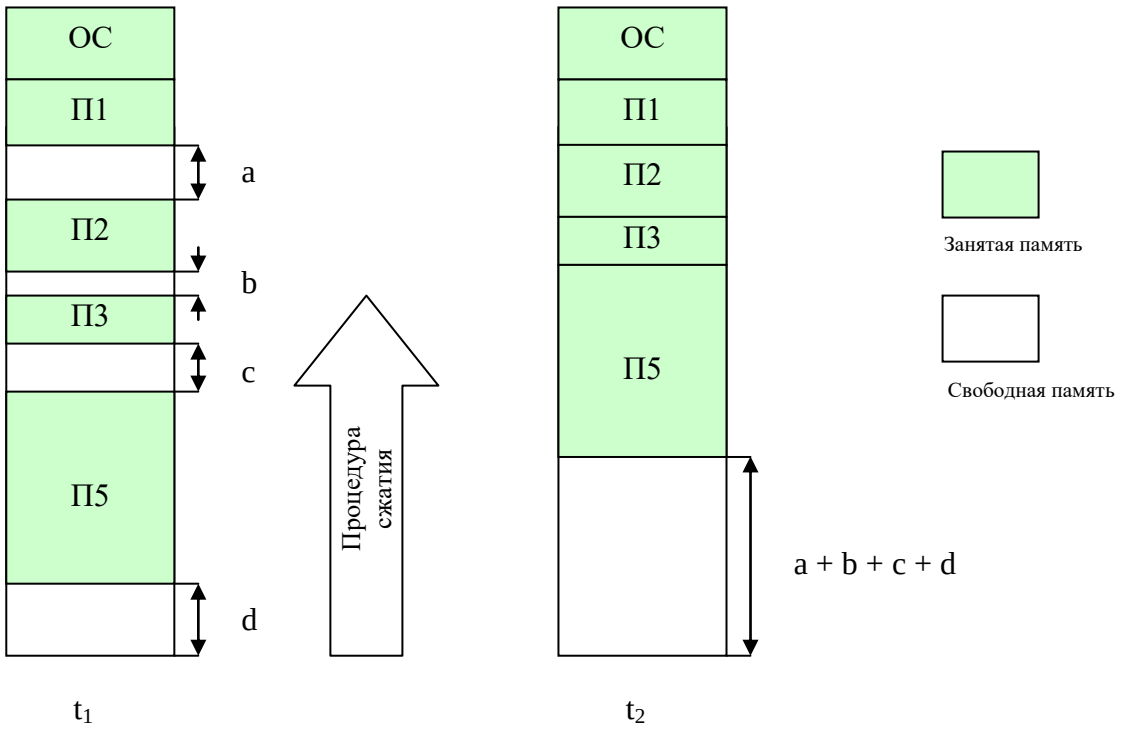


Рисунок 26 – Распределение памяти с перемещаемыми разделами

Так на рисунке представлены 4 свободных раздела, расположенных в разных участках памяти: a,b,c и d. Поле выполнения сжатия формируется один свободный участок памяти, по объему равный сумме предыдущих свободных разделов. Процедура сжатия позволяет более эффективно использовать доступный объем памяти, но приводит к снижению производительности вычислительной системы, так как требуются значительные интервалы времени на выполнение служебной операции. Данный метод распределения памяти применялся в ранних версиях OS/2.

3.3 Методы распределения памяти с подкачкой на жесткий диск

При большом количестве запущенных программ оперативной памяти может не хватить, что приведет к невозможности их выполнения. Одним из решений данной проблемы является возможность переноса части не исполняемых в текущий

момент процессов на жесткий диск, а при необходимости их выполнения динамически переносить их обратно в память.

Такая подмена физической оперативной памяти виртуальной, включающей ОЗУ и дисковую память, позволяет повысить уровень мультипрограммирования. Это обеспечивает увеличение количества одновременно выполняемых процессов, поскольку суммарный объем виртуальной памяти процессов может существенно превысить имеющийся объем физической. Виртуальным называется ресурс, предоставляющий пользователю или пользовательской программе свойства, которых в действительности не существует.

Механизм виртуализации памяти реализуется средствами операционной системы и процессора и включает решение следующих задач:

- размещение данных процессов в запоминающих устройствах разного типа, например, часть кода и данных программы – в ОЗУ, а другая – на дисковой памяти;
- перемещение по мере необходимости данных между памятью и диском;
- преобразование виртуальных адресов в физические.

Виртуализация памяти может быть реализована двумя способами:

- свопинг (swapping) или обычная подкачка – 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.

Недостатком свопинга является то, что при нем происходит перемещение всего раздела процесса, что приводит к передаче избыточной информации. Так как скорость обмена данными с дисковой памятью значительно медленнее, чем для оперативной, то это снижает производительность системы. Также операционная система не сможет загрузить для выполнения процесс, виртуальное адресное пространство которого превышает имеющуюся в наличии свободную память. В связи с этим, данный механизм управления памятью практически не используется в современных ОС.

Множество реализаций виртуальной памяти делится на три класса:

- страничная виртуальная память организует перемещение данных между памятью и диском страницами – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера;
- сегментная виртуальная память предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, сформированными с учетом смыслового значения данных;
- сегментно-страничная виртуальная память использует двухуровневое преобразование: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Данный способ управления памятью является комбинацией двух предыдущих подходов.

Для временного хранения сегментов и страниц на дисковой памяти выделяется специальная область или файл, которые во многих ОС по традиции продолжают называть областью или файлом свопинга.

3.3.1 Страничная организация памяти

В данном способе организации памяти виртуальное адресное пространство процесса делится на части фиксированного для данной системы размера, которые называются виртуальными страницами (virtual pages) (рисунок 27). Размер виртуального адресного пространства процесса не всегда кратен размеру страницы, поэтому последняя страница процесса дополняется фиктивной областью.

Физическая память компьютера также делится на блоки размером равным размерам виртуальных страниц. Данные блоки называется физическими страницами (или блоками, или кадрами). Размер страницы обычно выбирается равным степени двойки (512, 1024, 4096 байт и т. д.), что позволяет упростить механизм преобразования адресов.

Операционная система при запуске процесса загружает в оперативную память несколько его виртуальных страниц, которые будут использоваться в текущий момент (начальные страницы кодового сегмента и сегмента данных). При этом смежные виртуальные страницы могут быть размещены в несмежных физических

страницах. Далее управление передается коду загруженных страниц, а остальные загружаются по мере необходимости.

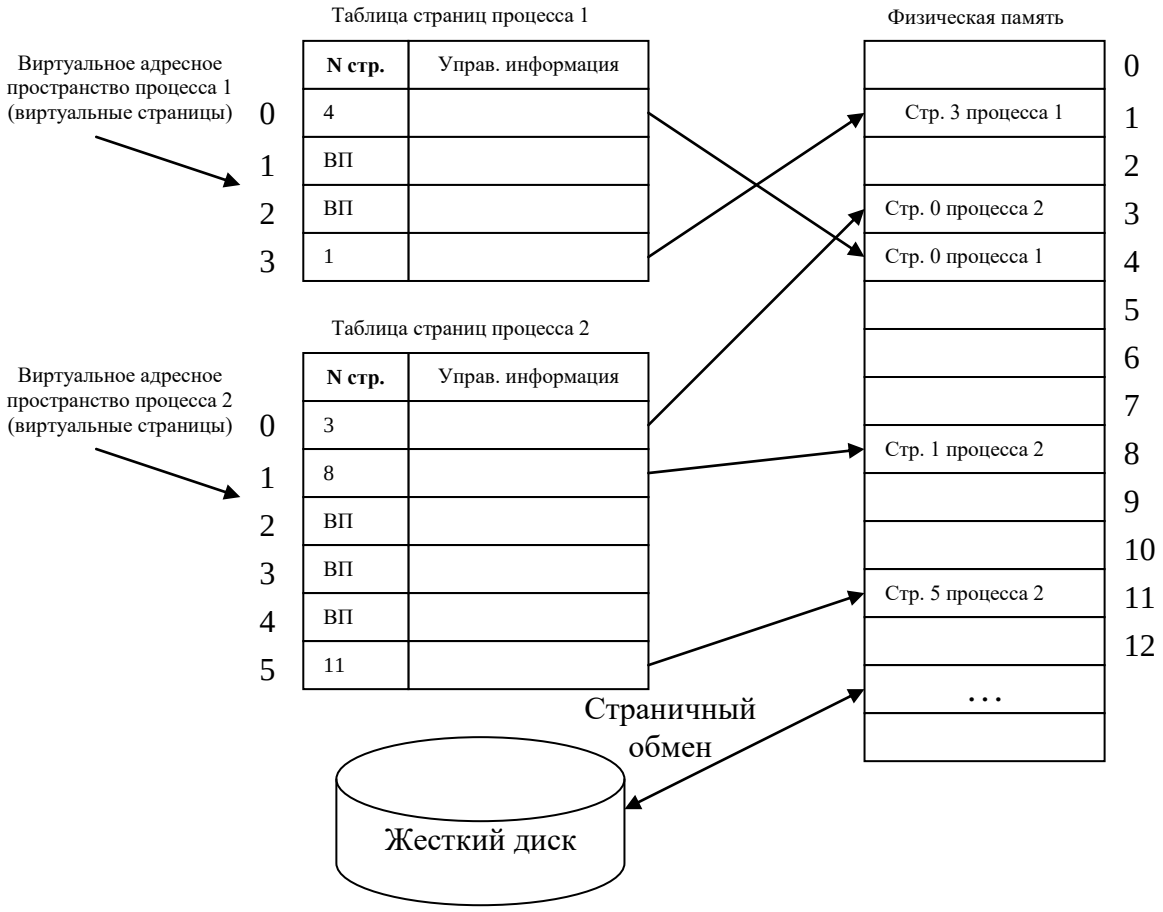


Рисунок 27 – Страничное распределение памяти

Для каждого процесса операционная система создает таблицу страниц, элементы которой определяют параметры всех виртуальных страницах процесса. Данная запись таблицы, называемая дескриптором страницы, содержит следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;
- признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

- признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Данные из таблицы дескрипторов страниц используются для принятия решения о необходимости перемещения страницы между памятью и диском, а также для определения физического адреса операндов и команд. Виртуальные адреса не передаются непосредственно на шину адреса процессора. Для этого используется диспетчер памяти (MMU – Memory Management Unit), осуществляющий преобразование виртуальных адресов в физические (рисунок 28). Диспетчер памяти в настоящее время обычно встраивается в микросхему процессора.

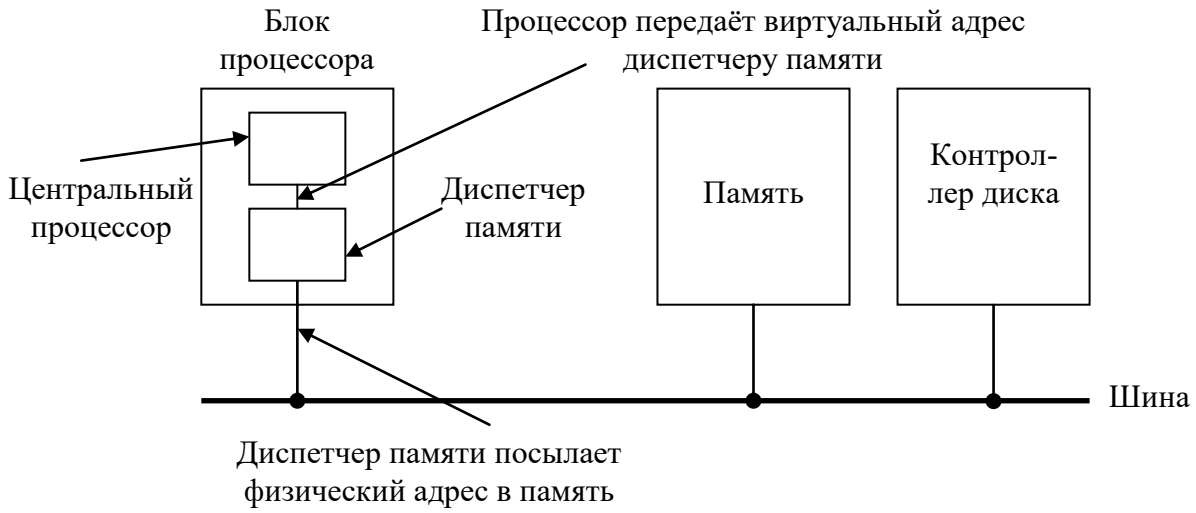


Рисунок 28 – Расположение и функции диспетчера памяти

При каждом обращении процессора к памяти первоначально выполняется поиск номера виртуальной страницы, содержащей требуемый виртуальный адрес. Данный номер используется как индекс таблицы страниц и позволяет извлечь необходимый дескриптор. На основе значения поля признака присутствия определяется, где сейчас находится виртуальная страница: в оперативной или внешней памяти. В первом случае, из дескриптора страницы извлекается адрес начала физической страницы (базовый адрес). Зная базовый адрес физической страницы и смещение виртуального адреса от начала страницы, рассчитывается физический адрес ячейки памяти.

Во втором случае, процессор инициирует страничное прерывание и передает управление ОС. Операционная система в ответ считывает отсутствующую в физической памяти страницу из диска и загружает в свободный участок ОЗУ. Если отсутствует свободный участок в оперативной памяти, то на основании принятой в данной системе стратегии замещения страниц ОС выгружает одну из страниц на внешнюю память. После обработки прерывания операционной системой, процессор повторно выполняет запрос к ячейке памяти.

Рассмотрим следующий пример: необходимо выполнить перемещение содержимого ячейки памяти с адресом «16394» в регистр процессора. Данная команда имеет следующий вид:

MOV REG, 16394

Исходные данные: размер страницы 4096 байт, состояние таблиц дескрипторов страниц такое, как представлено на рисунке 27. При выполнении данной команды виртуальный адрес 16394 передаётся в MMU. С учетом размера страниц, определяется, что данный адрес находится в третьей виртуальной странице и его смещение равно 10. Далее в третьей записи таблицы дескрипторов страниц считываются данные о том, что страница в текущий момент времени находится в физической памяти, начиная с адреса 4096. С учетом смещения 10 рассчитывается физический адрес данной ячейки памяти: 4096+10=4097.

Частота страничных прерываний, зависящая от размера страниц и алгоритма выбора страниц для выгрузки и загрузки, влияет на производительность системы. Не корректно выбранная стратегия замещения страниц может привести значительному увеличению времени, затрачиваемой ОС на подкачку страниц из оперативной памяти на диск и обратно. Это определяется следующими факторами:

- процесс, инициировавший загрузку страницы из внешней памяти в оперативную, на это время блокируется;

- операции ввода-вывода характеризуются большим временем доступа.

Результатом плохой стратегии принятия решений по замещению страниц может быть следующее: в текущий момент происходит обращение к странице на внешней памяти, которая была выгружена на предыдущем шаге. В связи с этим, в

алгоритмах замещения страниц используют критерии, смысл которых сводится к одному: на диск выталкивается страница, к которой дольше всего не будет обращений.

При страничной организации памяти есть 2 проблемы:

- таблица страниц может быть слишком большой. Это приведет к тому, что значительная часть оперативной памяти буде использоваться для хранения таблицы страниц, тем самым уменьшая свободный объем физической памяти;

- отображение страниц должно быть быстрым. Так как размеры страниц относительно небольшие, то используемые в небольшом интервале времени виртуальные адреса могут находится в различных страницах. Что приведет к частому определению базового адреса физических страниц.

Для решения первой проблемы используют многоуровневые таблицы памяти [20], при использовании которых в памяти находятся только части таблицы страниц.

Для решения второй компьютер снабжается внутренней кэш-памятью, использующейся для отображения виртуальных адресов в физические без обращения к таблице страниц в оперативной памяти. Это устройство называется буфером быстрого преобразования адреса (TLB – Translation Lookaside Buffer) или ассоциативной памятью. Обращение к внутреннему буферу TLB происходит гораздо быстрее, чем поиск дескриптора страницы в относительно медленной оперативной памяти.

В большинстве случаев, программы осуществляют большое число обращений к небольшому количеству страниц. Это приводит к тому, что в таблице страниц только малая доля записей используется интенсивно, а к остальной обращаются редко. Поэтому интенсивно используемая доля записей копируется в TLB, позволяя сократить время преобразования виртуального адреса в физический.

Когда происходит страничное прерывание и отсутствуют свободные участки в памяти, ОС должна определить страницу для выгрузки на диск. Если удаляемая страница была изменена за время своего присутствия в памяти, то необходимо обновить копию на диске. Однако, если страница не была модифицирована (например, она содержит текст программы), то её не нужно переписывать. Тогда

вновь загружаемая страница просто записывается поверх выгружаемой.

Хотя в принципе можно при каждом страничном прерывании выбирать случайную страницу для удаления из памяти, но производительность системы заметно повышается, когда предпочтение отдается редко используемой странице. Ниже описаны некоторые наиболее распространенные алгоритмы замещения страниц.

Оптимальный алгоритм.

В момент, когда принимается решение о выгрузке страницы, в памяти уже находится некоторое множество страниц. К некоторым страницам будет происходить обращение процессора в ближайшее время, а к другим не будет ссылок в течение следующих 10, 100 или даже 1000 команд. Рассматриваемый подход предполагает, что каждая страница характеризуется оценкой времени, через которое будет произведено к ней обращение. В этом случае, операционная система будет выгружать страницу с наибольшим значением данной характеристики.

Данный алгоритм нереализуем, так как операционной системе в момент страничного прерывания невозможно определить временные оценки ближайшего запроса к страницам в оперативной памяти. Несмотря на это, результаты моделирования данного алгоритма могут быть использованы для оценки качества других стратегий принятия решения о замещении страниц.

Алгоритм NRU – не использовавшаяся в последнее время страница.

Для сбора статистических данных о работе системы управления памятью в дескрипторе страниц используются два статусных бита:

- бит ***R*** (Referenced – обращения) устанавливается всякий раз, когда происходит обращение к странице (чтение или запись);

- бит ***M*** (Modified – изменение) устанавливается, когда содержимое страницы изменяется.

При этом данные биты могут поддерживаться аппаратными средствами процессора или реализовываться на уровне операционной системы.

Данный алгоритм использует биты ***R*** и ***M*** для принятия решения о замещении страницы. При запуске процесса оба страничных бита установлены на 0.

Периодически (например, при каждом прерывании по таймеру) операционная система сбрасывает биты ***R*** дескрипторов страниц. Это позволяет определить страницы, к которым давно не было обращения.

При возникновении страничного прерывания ОС делит все страницы на четыре класса на основании текущих значений битов обращения и модификации:

- класс 0: не было обращений и изменений;
- класс 1: не было обращений, страница изменена;
- класс 2: было обращение, страница не изменена;
- класс 3: произошло и обращение, и изменение.

Алгоритм NRU (Not Recently Used) удаляет случайную страницу в непустом классе с наименьшим номером. Достоинством данного алгоритма является то, что он простой в реализации и дает приемлемую оценку производительности.

Алгоритм FIFO – первым прибыл – первым обслужен.

Операционная система фиксирует время перемещения страницы в оперативную память и формирует список: первая страница в списке является старейшей, а страницы в хвосте списка попали в него совсем недавно. Когда происходит страничное прерывание, то выгружается из памяти страница в начале списка, а загружаемая добавляется в его конец. Данный алгоритм не используется, так как он может удалить наиболее часто вызываемую страницу.

Алгоритм «вторая попытка».

Представляет собой модификацию алгоритма FIFO. Когда происходит страничное прерывание, то у самой «старейшей» страницы проверяется дополнительно бит обращения ***R***. Если он равен 0, т.е. к странице долго не обращались, то страница выгружается из памяти. Если же было обращение к странице, то она не выгружается, а перемещается в конец очереди (становится самой «молодой»), сбрасывая бит ***R*** дескриптора данной страницы.

Алгоритм «часы».

Недостатком предыдущего алгоритма является то, что происходит перемещение элементов списка страниц, что приводит к дополнительным временным затратам на перемещение данных в оперативной памяти. Поэтому

эффективнее хранить страничные блоки в кольцевом списке наподобие часов, как показано на рисунке 29. Стрелка указывает на старейшую страницу.

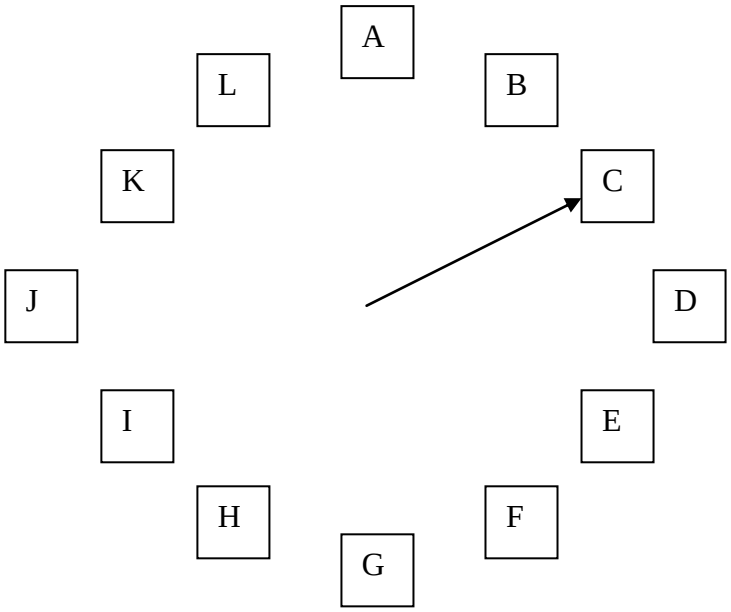


Рисунок 29 – Кольцевой список в алгоритме «часы»

Когда происходит страничное прерывание, проверяется страница, на которую указывает стрелка. Если ее бит ***R*** равен 0, то страница выгружается и на ее место в часовом круге встает вновь загруженная страница. При этом стрелка сдвигается вперед на одну позицию. Если же бит ***R*** равен 1, то он сбрасывается, стрелка перемещается к следующей странице. Этот процесс повторяется до тех пор, пока не находится та страница, у которой бит обращения равен нулю.

Алгоритм LRU – страница, не использовавшаяся дольше всего.

Данный алгоритм базируется на следующей идее: страницы, к которым происходит большое количество обращений, наиболее вероятно, также часто будут использоваться в следующие моменты времени. И наоборот, страницы с малым количеством обращений будут редко использоваться в течение долгого времени. Реализация данного алгоритма заключается в следующем: когда происходит страничное прерывание, то выгружается из памяти страница, которая не использовалась дольше всего. Такая стратегия замещения страниц называется LRU (Least Recently Used – «вытеснение давно неиспользуемых»).

Для полного осуществления алгоритма LRU необходимо поддерживать связный список всех содержащихся в памяти страниц, где последняя использовавшаяся страница находится в начале списка, а та, к которой дольше всего не было обращений – в конце. Данный алгоритм труднореализуем в связи с тем, что список должен обновляться при каждом обращении к памяти. Поиск страницы, ее удаление, а затем вставка в начало списка – это операции, поглощающие очень много времени, даже если они выполняются аппаратно (если предположить, что необходимое оборудование можно сконструировать). Способы реализации данного алгоритма описаны в работе Э. Таненбаума [20], однако из-за необходимости аппаратной поддержки разработчики операционных систем редко им пользуются.

Алгоритм «старение».

Разновидностью схемы LRU является алгоритм NFU (Not Frequently Used – редко использовавшаяся страница). Для него требуются счетчики, связанные с каждой страницей в памяти. Периодически операционная система сканирует все дескрипторы страниц, находящихся в памяти. Если бит обращения ***R*** страницы равен 1, то увеличивается его счетчик. Таким образом, счетчики содержат количество обращений к соответствующей странице. При страничном прерывании для выгрузки выбирается страница с наименьшим значением счетчика.

Недостатком алгоритма NFU является то, что он подсчитывает общее количество обращений, не учитывая её динамику. Например, в многоходовом компиляторе страницы, которые часто использовались во время первого этапа, могут все еще иметь высокое значение счетчика и на более поздних этапах, когда к этим же страницам обращения происходят редко. Модификация алгоритма позволяет исправить данный недостаток:

- каждый счетчик сдвигается вправо на один разряд перед прибавлением бита ***R*** (фактически значение счетчика уменьшается в 2 раза);
- бит ***R*** добавляется старшему разряду счетчика.

В таблице 3 продемонстрированы результаты работы видоизмененного алгоритма, известного под названием «старение» (aging). На первом такте было обращение к страницам 0, 2, 4 и 5 (их старшие разряды равны 1). На следующем

такте происходит сдвиг содержимых счетчиков и добавление соответствующих битов обращения. Остальные четыре колонки таблицы изображают шесть счетчиков после следующих четырех тактов.

В случае страничного прерывания, выгружается из памяти страница, чей счетчик имеет наименьшее значение. Страница, к которой не было последние такты обращения, будет иметь счетчик с нулями в старших разрядах, что соответствует меньшему значению. Например, на 4 такте будет выгружаться 3 страница, так как значение его счетчика наименьшее среди остальных.

Таблица 3 – Пример работы алгоритма «старение»: в строках – 0-5 страницы памяти; в столбцах – биты ***R*** для страниц

С	**Значения бита R для 5 страниц**				
	Такт 0: 101011	**Такт 1: 110010**	**Такт 2: 110101**	**Такт 3: 100010**	**Такт 4: 011000**
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

Алгоритм «рабочий набор».

В обычных системах со страничной подкачкой во время запуска нового процесса необходимые для его выполнения страницы отсутствуют в памяти. При попытке процессора выбрать первую команду программы происходит страничное прерывание, при обработке которого операционная система запускает процесс замещения страниц. Далее процессу необходимы страницы, содержащие глобальные переменные и стек, что приводит к новым страничным прерываниям. В результате за некоторое время в памяти окажется минимальный набор процесса, позволяющее запустить его исполнение. После этого количество исключительных ситуаций с обращением к страницам, отсутствующим в оперативной памяти, будет

минимально. Данный подход называется замещением страниц по запросу (demand paging), так как страницы загружаются в память по требованию.

Обычно процессы во время выполнения обращаются к небольшому количеству своих страниц. Множество страниц, которое процесс использует в данный момент, называется рабочим набором. Основная идея данного алгоритма замещения страниц заключается в том, чтобы выбрать страницу, не включенную в рабочий набор, и выгрузить ее. Каждый дескриптор страницы содержит, кроме бита обращения ***R,*** дополнительное поле: приблизительное время последнего использования ***T***.

Алгоритм работает следующим образом. Аппаратное обеспечение устанавливает биты ***R*** и ***M*** соответственно при обращении и модификации страницы. А операционная система периодически сбрасывает бит ***R*** (чтобы очистить устаревшие данные об обращениях к страницам). При каждом страничном прерывании сканируется таблица страниц и для записей, у которых выставлен бит ***R***, записывается текущий интервал времени процессора, который был выделен процессу с момента его старта. Страница может быть выгружена, если ***R*** равен нулю и разность текущего времени и времени последнего обращения («возраст») большего заданной величины ***t***. Это означает, что данная страница уже не относится рабочему набору. Если проверена вся таблица, а кандидат на удаление не найден, это означает, что все страницы входят в рабочий набор. В этом случае, если были найдены одна или больше страниц с битом ***R*** = 0, удаляется та из них, которая имеет наибольший возраст.

В случае, когда в текущем такте было обращение ко всем страницам, тогда страница на выгрузку выбирается случайным образом, отдавая предпочтение страницам без модификации (чтобы не пришлось переписывать копию на внешней памяти). Данный алгоритм является очень трудоемким, так как при каждом поиске страницы на выгрузку приходится проверять всю таблицу страниц до тех пор, пока не определится местоположение подходящего кандидата.

Алгоритм WSClock.

Данный подход является модификацией предыдущего. Для его применения необходим кольцевой список (рисунок 29). Первоначально этот список пустой. При загрузке страницы, её дескриптор добавляется в конец списка, формируя кольцо. Каждый дескриптор страницы, кроме битов ***R*** и ***M***, содержит поле «время последнего использования» аналогично базовому алгоритму «рабочий набор».

При каждом страничном прерывании первой проверяется та страница, на которую указывает стрелка. Если для текущей записи бит ***R*** равен 1, это значит, что страница использовалась в течение последнего такта часов, и поэтому не является кандидатом на удаление. В этом случае, бит ***R*** сбрасывается, стрелка передвигается на следующую запись и для нее повторяются те же действия.

Если в момент проверки бит ***R*** равен нулю и время последнего использования больше заданной величины ***t***, то проверяется бит наличия изменений ***M***. Если страница была неизменной, то страница выгружается из памяти. Если страница была изменена, то она помечается как планируемая для копирования, а стрелка сдвигается на следующий элемент.

Если стрелка часов совершила полный круг и не нашла кандидата на выгрузку, то возможны два варианта:

1) была запланирована операция выгрузки страницы на внешнюю память;

2) не было запланировано ни одной страницы на выгрузку.

В первом случае предполагаем, что за время оборота по кругу какая-нибудь модифицированная страница сохранится на диске и станет неизменным. Таким образом, будет выбрана первая попавшаяся страница без изменений с битом ***R*** равным 0.

Во втором случае получается, что все страницы относятся к рабочему набору. Тогда выгружается случайная страница без изменений.

Наилучшими характеристиками среди рассмотренных обладают алгоритмы «старение» и WSCIock. Оба обеспечивают хорошую постраничную подкачку и могут быть реализованы при небольших вычислительных затратах.

Недостатки страничного распределения памяти – размеры страниц и частота страничных прерываний сильно влияют на производительность компьютера. Также

все данные в памяти находятся перемешенными друг с другом, что значительно усложняет, в частности, реализацию механизмов защиты для них.

3.3.2 Сегментная организация памяти

При страничной организации виртуальное адресное пространство процесса делится на равные части, без учета смыслового значения данных. В одной странице могут оказаться и команды, и инициализируемые переменные, и массив исходных данных программы. Подобный подход не позволяет осуществить дифференцированный доступ к разным частям программы, который бывает полезным во многих случаях, например, чтобы запретить случайную или преднамеренную модификацию сегмента кода. Но при этом для сегмента данных необходимо разрешить операцию модификации.

Также разбиение виртуального адресного пространства на сегменты по типу данных делает возможным совместное использование фрагментов программ разными процессами. Свойство повторной входимости кода, которое позволяет одновременно использовать его несколькими процессами, называется реентерабельностью[16]. При выполнении реентерабельного кода процессы не изменяют его, поэтому в память достаточно загрузить только одну её копию. Тогда коды этой подпрограммы могут быть оформлены в виде отдельного сегмента и включены в виртуальные адресные пространства обоих процессов, которые отображаются на один и тот же сегмент в физической памяти. На рисунке 30 приведен пример, когда дескрипторы сегментов 1и 2 соответственно процессов A и B указывают на один сегмент памяти. Это позволяет экономить используемую оперативную память.

Виртуальное адресное пространство процесса делится на части, называемые сегментами. Размеры сегментов разные и зависят от смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Деление виртуального адресного пространства на сегменты осуществляется компилятором, в соответствии с принятыми в системе

соглашениями. Максимально возможный размер сегмента определяется разрядностью виртуального адреса. Так для 32-разрядного процессора он равен 2^{32}= 4 Гбайт. Максимальный размер виртуального адресного пространства процесса представляет собой набор из N виртуальных сегментов, каждый размером по 4 Гбайт.

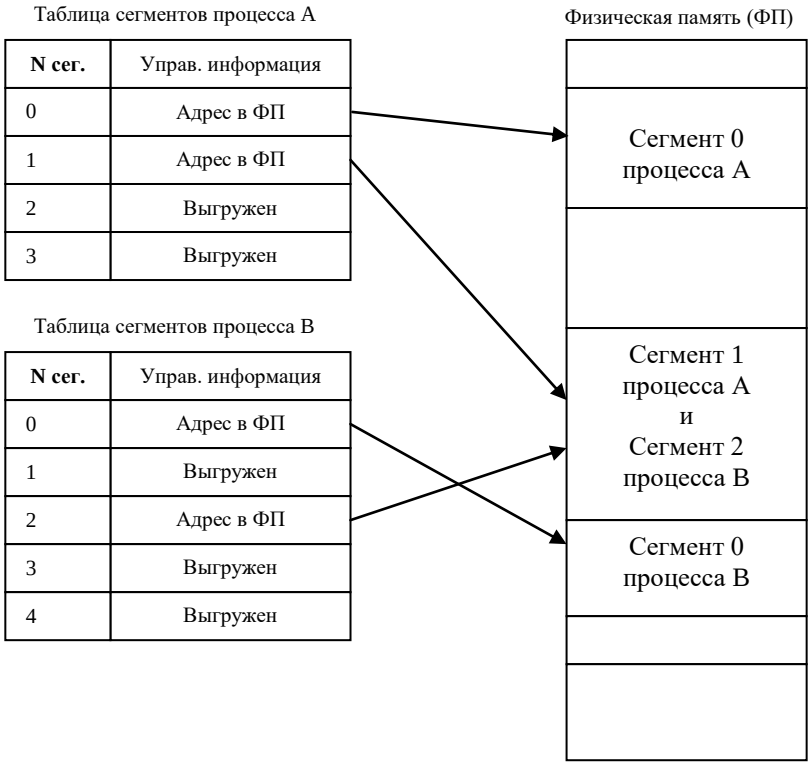


Рисунок 30 – Распределение памяти сегментами

При загрузке процесса в оперативную память помещается только часть его сегментов, полная копия виртуального адресного пространства находится в дисковой памяти. Для каждого загружаемого сегмента ОС подбирает непрерывный участок свободной памяти достаточного размера. При этом смежные в виртуальной памяти сегменты процесса могут занимать в оперативной памяти разные участки. При обращении по виртуальному адресу, относящемуся к отсутствующему в оперативной памяти сегменту, происходит прерывание. Операционная система блокирует данный активный процесс и запускает на выполнение следующий процесс из очереди, параллельно организуя загрузку нужного сегмента с внешней памяти.

Во время создания и загрузки процесса в оперативную память ОС создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается:

- базовый физический адрес начала сегмента;
- размер сегмента;
- правила доступа к сегменту;
- признаки модификации, присутствия и обращения к данному сегменту, а также некоторая другая информация.

Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружен в единственном экземпляре.

Недостатками сегментного распределения памяти являются:

1) более медленное преобразование виртуального адреса в физический в сравнении со страничным распределением;

2) избыточное копирование данных из оперативной памяти во внешнюю и обратно, так как во многих случаях нужна только часть информации сегмента. Но приходится загружать весь сегмент;

3) фрагментация, которая возникает из-за разного значения размеров сегментов. В процессе работы системы в памяти образуются небольшие участки свободной памяти, в которые не может быть загружен ни один сегмент. Суммарный объем, занимаемый подобными фрагментами, может составить существенную часть общей памяти системы, приводя к ее неэффективному использованию.

Достоинством сегментной организации памяти является возможность задания различных прав доступа процесса к его сегментам. Например, один сегмент данных, содержащий исходную информацию для приложения, может иметь права доступа «только чтение», а сегмент данных, представляющий результаты – «чтение и запись».

3.3.3 Сегментно-страничная организация памяти

Данный метод представляет собой комбинацию страничного и сегментного механизмов управления памятью, который совмещает достоинства обоих подходов.

Так же как и при сегментной организации памяти, виртуальное адресное пространство процесса разделено на сегменты. Это позволяет определять разные права доступа к разным частям программы.

Перемещение данных между памятью и диском осуществляется не сегментами, а страницами. Для этого каждый виртуальный сегмент и физическая память делятся на страницы равного размера, что позволяет более эффективно использовать память, сократив до минимума фрагментацию.

3.4 Кэширование данных

Кэш-память – это способ совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который за счет динамического копирования в «быстрое» запоминающее устройство наиболее часто используемой информации из «медленного» позволяет, с одной стороны, уменьшить среднее время доступа к данным, а с другой стороны, экономить более дорогую быстродействующую память [16].

Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств – «быстрое» запоминающее устройство. В современных процессорах кэш-память размещается в кристалле процессора и представляет собой «быструю» статическую память. Применение кэш позволяет повысить производительность системы за счет того, что в компьютере одним из «узких» мест являются циклы обращения к памяти, которые выполняются значительно медленнее, чем команды процессора. Обращение к копиям наиболее часто используемых данных в быстрой памяти позволяет сократить задержки в работе процессора. Для программиста кэш-память прозрачна - он не может указать, чтобы данные были размещены ней.

Содержимое кэш-памяти представляет собой совокупность *записей* обо всех загруженных в нее элементах данных из основной памяти. Каждая запись об элементе данных включает в себя:

- значение элемента данных;
- служебную информацию (тэг), позволяющую определить, находится ли запрашиваемые процессором по заданному адресу данные в кэш-памяти;
- дополнительную информацию, используемую для реализации алгоритма замещения данных в кэше и обычно включающего признак модификации и признак действительности данных.

На рисунке 31 продемонстрирована схема работы кэш-памяти.

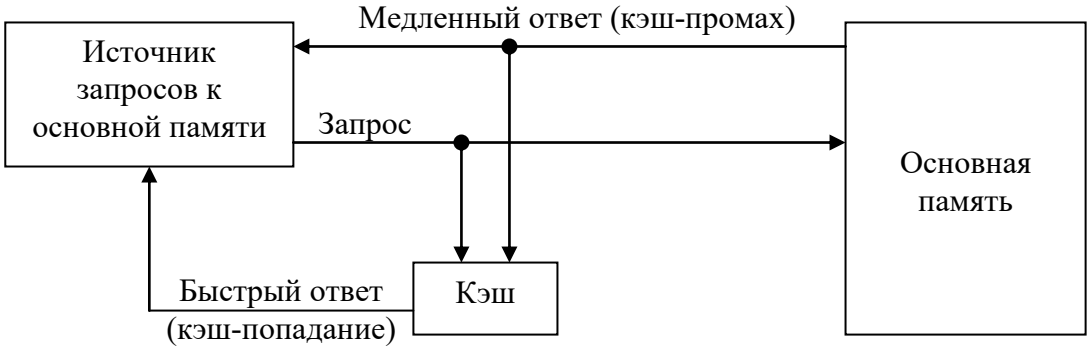


Рисунок 31 – Схема функционирования кэш-памяти

Перед обращением к основной памяти за данными процессор передает запрос со значением физического адреса нужной ячейки контроллеру кэш-памяти, который просматривает содержимое кэш с целью определения нахождения копии нужных данных. Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому, которое взято из запроса, например, части физического адреса. На рисунке 32 продемонстрирован обобщенный пример поиска данных в ассоциативной кэш-памяти.

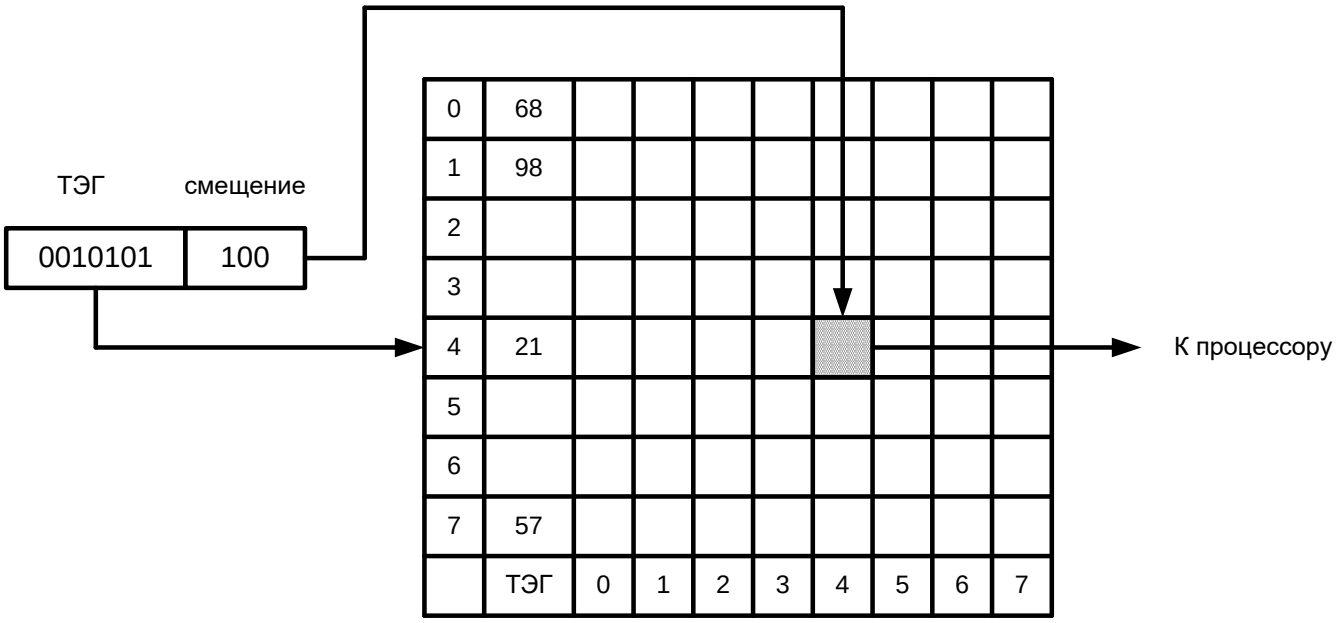


Рисунок 32- Схема работы ассоциативной кэш-памяти

Контроллеру кэш-памяти поступает 10-разрядный физический адрес, из которого выделяют поля тэг и смещение. Далее контроллер сканирует строки и просматривает значение их поля тэг на совпадение с входным значением. Если будет найден соответствующий тэг, то это означает, что данные находятся в кэш-памяти в данной строке. Индекс элемента в данной строке определяется значением смещения, выделенного из адреса. Если в результате сканирования нужный тэг не найден, то это значит, что данных нет в кэш-памяти. В примере старшие 7 разрядов адреса образуют поле тэг со значением 21. Как видно из рисунка, запрашиваемые данные находятся в кэш-памяти в 4-ой строке со смещением 4.

Если данные обнаруживаются в кэш-памяти (такая ситуация называется кэш-попаданием), то они считываются из нее и передаются источнику запроса. Если же запрашиваемые данные отсутствуют в кэш-памяти (кэш-промах), то они считываются из основной памяти. Одновременно копируется блок из основной памяти с запрашиваемой ячейкой в кэш-память на случай, если данные будут запрашиваться в ближайшее время.

Использование кэш-памяти имеет смысл только при высокой вероятности кэш-попадания. Вероятность нахождения запрашиваемых данных в кэш-памяти зависит от множества факторов, таких как объем кэшируемой памяти, алгоритм

замещения данных в кэше, особенности исполняемой программы, времени её работы, уровня мультипрограммирования и других особенностей вычислительного процесса. В современных компьютерах процент кэш-попаданий свыше 90 %. Высокое значение вероятности нахождения данных в кэш-памяти определяется двумя свойствами: пространственной и временной локальностью.

Временная локальность предполагает, что если произошло обращение к данным по заданному адресу, то следующее обращение с высокой вероятностью будет по тому же адресу. Пространственная локальность заключается в том, что если произошло обращение по определенному адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним ячейкам памяти.

Данные свойства позволяют загружать копии наиболее часто используемых данных в более быструю память небольшого объема, что обеспечивает высокую производительность при приемлемой стоимости.

Если при выполнении запроса к информации их не оказалось в кэш-памяти, то процессор должен считать блок с искомыми данными из ОЗУ и разместить в кэш. Так как размер кэш-памяти небольшой, то она может быть полностью заполненной. Тогда необходимо выбрать блок в кэш-памяти для вытеснения. Вытеснение заключается в объявлении свободным области кэш-памяти (сброс бита действительности) при условии, если данные не были модифицированы, иначе дополнительно осуществляется копирование данных в основную память. Используемый алгоритм вытеснения данных кэш-памяти влияет на производительность компьютера. При этом должны выполняться два требования:

- алгоритм должен быть быстрым, чтобы не замедлять работу кэш-памяти;
- обеспечивать максимально возможную вероятность кэш-попаданий.

Примерами алгоритмов замещения, которые определяют претендентов на выгрузку из кэш-памяти, являются:

- случайный выбор блока;
- выбор в последовательном порядке (FIFO);
- выбор блока, который дольше всего не использовался (LRU).

Наличие в компьютере двух копий данных (в основной памяти и в кэше) определяет задачу их согласования. Если происходит изменение данных в кэш, то его значение в основной памяти становится недостоверным. Существуют следующие методы решения данной задачи:

1 *Сквозная запись (write through).* При изменении данных запись производится одновременно в кэш-память и в оперативную память. Так как запись в оперативную память осуществляется долго относительно кэш, то это приводит к снижению производительности вычислительной системы.

2*Обратная запись (write back).* При модификации данных информация изменяется только в кэш-памяти. Основная идея данного подхода заключается в том, что во время выполнения программы данные в кэш много раз меняются, так и не попав в оперативную память. При изменении записи в кэш-памяти делается специальная отметка (признак модификации), которая указывает на то, что при вытеснении этих данных из кэша необходимо переписать их в основную память. Данный способ является более эффективным, так как сокращает количество медленных операций записи в оперативную память.

Современные компьютеры характеризуются более сложной структурой организации кэш-памяти, заключающейся в том, что они содержат несколько кэшти разных уровней: 1 и 2 уровня. При этом схема взаимодействия между кэш 1-го и 2-го уровня похожа на ранее рассмотренные принципы взаимодействия кэш-памяти и основной памяти:

- кэш 1-го уровня является более быстрой, чем 2-го уровня;

- кэш 1-го уровня по объему меньше 2-го уровня.

Поэтому наиболее часто используемые данные должны располагаться в кэш 1-го уровня. При этом существует несколько схем соотношения данных между ними:

- кэш 1-го уровня содержит копию данных 2-го уровня;

- кэш 1-го и 2-го уровней содержат разные данные, тем самым увеличивают общий объем кэша.

Также кэш-памть может быть разделена по типу хранимых данных: кэш-память данных или данных. Это увеличивает производительность системы, так как

позволяет процессору считать одновременно как команду, так и необходимые операнды.

3.5 Контрольные вопросы

1 Может ли прикладной процесс использовать системную часть виртуальной памяти?

2 Какое из этих двух утверждений верно?

A) все виртуальные адреса заменяются на физические во время загрузки программы в оперативную память;

B) виртуальные адреса заменяются на физические во время выполнения программы в момент обращения по данному виртуальному адресу.

3 Распределение памяти перемещаемыми разделами основано на применении процедуры сжатия. Имеет ли смысл использовать данную процедуру при страничном распределении? А при сегментном?

4 На что влияет размер страницы?

5 У маленького компьютера четыре страничных блока. Во время первого такта часов биты R равны 0111 (у страницы 0 бит R равен 0, у остальных - 1). Во время последующих тактов часов биты R принимают значения 1011,1010, 1101, 0010, 1010, 1100 и 0001. Считая, что используется алгоритм старения с 8-разрядным счетчиком, напишите четыре значения, которые примет счетчик после последнего тика.

6 В кэше хранятся данные, которые наиболее активно используются в последнее время. Каким образом система определяет, какие данные должны быть загружены в кэш?

7 Как обеспечивается согласование данных в кэше с помощью методов обратной и сквозной записи?

8 Перечислите недостатки страничной организации памяти.

9 Перечислите недостатки сегментной организации памяти.

10. Какое назначение имеет бит ***R*** в дескрипторе страниц?

11. В чем недостаток алгоритма замещения страниц FIFO?

12. Что называется рабочим набором процесса? Как он может использоваться для замещения страниц?

13. Сегментная или страничная организация памяти позволяет реализовать многоуровневые права доступа к участкам памяти?

14. Какой тип организации памяти склонен к фрагментации – сегментная или страничная? Поясните почему?

15. Поясните, что такое виртуальная память?

16. Поясните отличия между физическим и виртуальным адресами?

17. Как влияет кэш-память на производительность вычислительной системы? Обоснуйте ответ.

18. Как работает ассоциативная кэш-память?

4 Система ввода-вывода информации

4.1 Принципы аппаратного обеспечения ввода-вывода

Возможности компьютера во многом определяются его внешними устройствами. Многообразие внешних устройств делает актуальной задачу обеспечения обмена данными между приложениями и периферийными устройствами компьютера. Данную задачу в современных вычислительных системах выполняет операционная система, которая управляет всеми устройствами ввода-вывода.

Все внешние устройства ввода-вывода делятся на две класса: блочные и символьные устройства. Блочными называются устройства, хранящие информацию в виде блоков фиксированного размера, каждый из которых определяется адресом. Запись и чтение данных из устройства данного класса осуществляется фиксированными блоками. Примерами подобных устройств являются жесткие диски, флэш-память.

Символьное устройство представляется как поток символов без наличия блочной структуры. Также отсутствует адресация блоков данных, нет средств позиционирования в нужной точке. Примером подобных устройств являются принтеры, сетевые карты, мыши. Однако некоторые устройства нельзя отнести к данным обширным классам, например, часы, которые отсчитывают интервалы времени и формируют прерывания.

Кроме того, необходимо отметить, что внешние устройства компьютера отличаются по скорости обмена данными. Скорость работы устройств ввода-вывода колеблется от 10 байт в секунду да десятков гигабайт в секунду. Данное обстоятельство определяет необходимость различных способов взаимодействия прикладных программ и периферийных устройств.

В большинстве случаев, устройство ввода-вывода включает механическую и электронную части. Электронный компонент устройства называется контроллером устройства или адаптером, который принимает вид печатной платы, вставляемой в разъём. Контроллер представляет собой отдельный модуль. Принцип модульной организации позволяет компьютеру взаимодействовать с большим количеством разнообразных внешних устройств – достаточно включить в состав системы соответствующий контроллер.

Интерфейс низкого уровня между устройством и контроллером обеспечивает конвертирование последовательного потока битов в блок байтов и выполнение коррекции ошибок. После чего, этот блок байтов уже обслуживает операционная система. Для взаимодействия с внешним устройством используются набор управляющих сигналов, который является в большинстве случаев индивидуальным для каждого класса устройств. Контроллер внешнего устройства осуществляет формирование управляющих сигналов, предоставляя вычислительному ядру общий интерфейс взаимодействия. Для этого контроллер содержит регистры, с помощью которых с ним взаимодействует процессор. Данные регистры получили название порты. В определенные регистры процессор записывает данные и команды, из других считывает информацию о состоянии внешнего устройства. Кроме того, в

контроллере может быть буфер данных, из которого операционная система может читать или записывать туда большие массивы данных (например, видеопамять).

Доступ к управляющим регистрам и буферам данных устройств ввода-вывода осуществляется двумя основными способами:

- каждому управляющему регистру назначается уникальный номер (8-или 16-разрядное целое число), использующийся как адрес. В этом случае процессор для записи и чтения использует специальные команды ввода (IN) и вывода (OUT). При такой схеме адресные пространства ОЗУ и устройств ввода-вывода не пересекаются (рисунок 33, а). Такая организация адресного пространства ввода-вывода характерна для старых компьютеров;

- отображение всех управляющих регистров периферийных устройств на адресное пространство памяти (рисунок 33, б). Для взаимодействия с портами внешних устройств используются обычные команды записи и чтения, как для ячеек оперативной памяти.

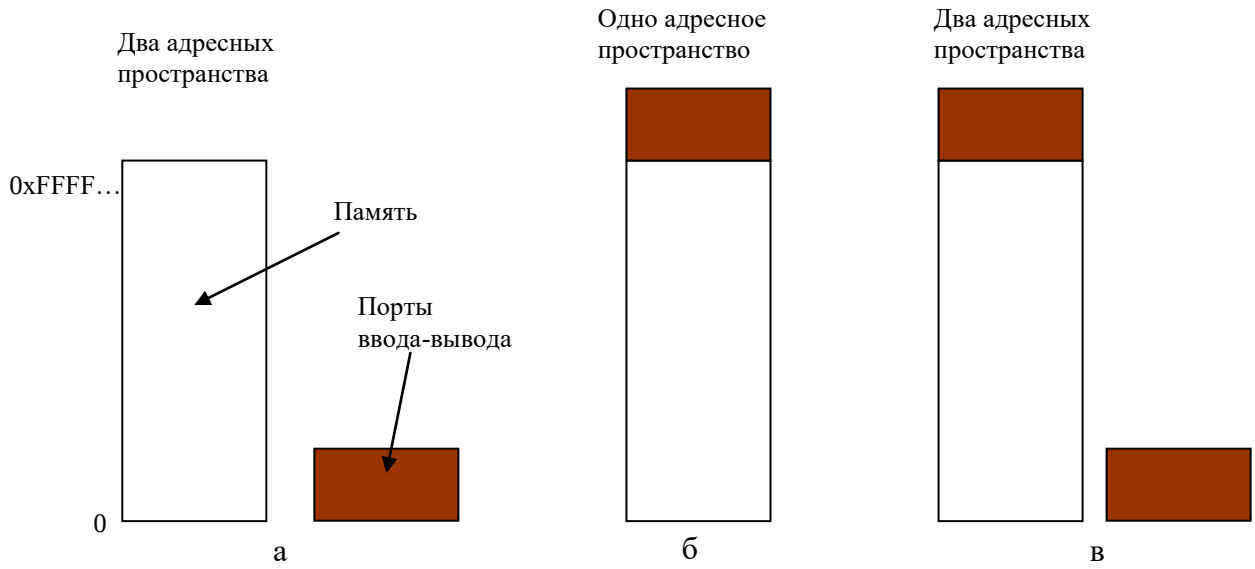


Рисунок 33 – Раздельные адресные пространства (а); отображаемый на адресное пространство ввод-вывод (б); гибрид (в)

Также возможно применение смешанной схемы. Достоинствами ввода-вывода, отображаемого на адресное пространство памяти, являются:

- для обращения к устройствам ввода-вывода не нужны специальные команды, позволяет использовать большое количество видов адресации, что упрощает написание программ;
- не требуется специального механизма защиты от пользовательских процессов, обращающихся к устройствам ввода-вывода, так как область памяти с портами исключается из адресного пространства пользователей.

Недостатками ввода-вывода, отображаемого на адресное пространство, являются:

- регистры ввода-вывода нельзя кэшировать, так как в этом случае мы бы никогда не узнали состояния портов, поэтому увеличивается сложность управления избирательным кэшированием;
- все устройства ввода-вывода должны изучать все обращения к памяти центрального процессора, что в схемах с более чем одной шиной можно сделать только с помощью фильтрации адресов специальной микросхемой, что и сделано ещё на базе процессора Pentium I.

Обмен данными с внешним устройством с помощью команд ввода-вывода (IN и OUT) через порты контроллера называется программно-управляемым. Недостатком данного способа является, что при взаимодействии с медленными устройствами процессор часто выполняет циклы ожидания готовности внешнего устройства. Это приводит к снижению производительности системы.

Большое значение для осуществления ввода-вывода с внешними устройствами имеет система прерываний компьютера. Данный способ обмена данными эффективен при взаимодействии с медленными внешними устройствами. Прерывание (англ. interrupt) – сигнал, сообщающий процессору о совершении какого-либо асинхронного события. Обычно при помощи данного сигнала внешние устройства сообщают о возникновении события, требующего обмена данными. При этом процессор приостанавливает выполнение текущей последовательности команд, и управление передаётся подпрограмме (обработчику прерываний), которая выполняет работу по обработке события. После завершения обработки прерывания

процессор продолжает выполнение прерванной программы. Схема работы системы прерываний представлена на рисунке 34.

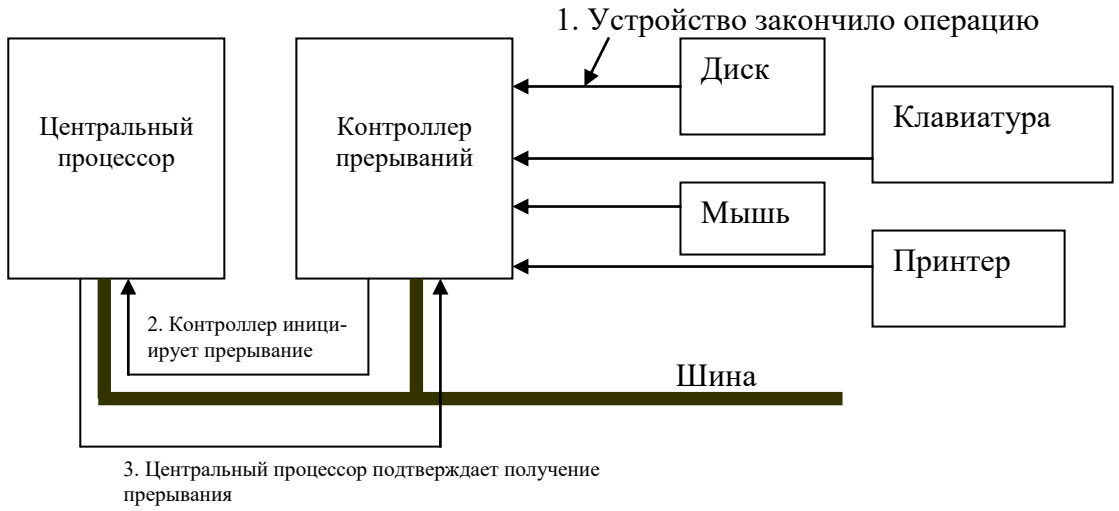


Рисунок 34 – Схема прерываний в компьютере

В связи со сложностью организации обработки прерываний в систему с большим количеством внешних устройств включают контроллер прерываний (обычно называется программируемый контроллер прерываний, ПКП). Данное устройство реализует большую часть функций, необходимых для системы прерываний, например, определения приоритета запросов от внешних устройств. Когда внешнему устройству необходимо осуществить обмен данными с ядром компьютера, тогда оно инициирует прерывание, которое поступает на контроллер ПКП. Контроллер прерываний определяет приоритет запросов от внешних устройств, формирует сигнал к процессору, чтобы он приостановил выполнение текущей программы и ожидает подтверждения от него. При получении сигнала от процессора о начале обработке данного прерывания, контроллер ПКП передает на системную шину номер устройства, требующего к себе внимания. Номер, выставленный на системную шину, используется в качестве индекса в таблице векторов прерываний (справедливо для компьютеров, использующих векторную систему прерываний). Каждая запись данной таблицы содержит адрес на подпрограмму обработки прерывания. По полученному номеру из таблицы

векторов прерываний извлекается новое значение счетчика команд и запускается выполнение обработчика прерывания.

Для того, чтобы сохранить результаты выполнения основной программы, процессор перед запуском обработчика прерываний, сохраняет содержимое регистров в стек. После завершения обработки прерывания, процессор восстанавливает содержимое регистров и продолжает работу, как будто и не было прерывания.

Процессор Pentium поддерживает векторную схему прерываний, позволяющую вызвать 256 процедур обработки прерываний. Соответственно, таблица векторов прерываний имеет 256 элементов, которые в реальном режиме работы процессора содержат дальние адреса (CS:IP) этих процедур, а в защищенном режиме – дескрипторы. Контроллер прерываний в большинстве аппаратных платформ на основе процессоров Pentium реализует механизм опрашиваемых прерываний, поэтому общий механизм компьютера носит смешанный векторно-опрашиваемый характер. Прерывания, которые обрабатывает Pentium, делятся на следующие классы [16]:

- аппаратные (внешние) прерывания – 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. Схема обработки маскируемых прерываний фактически была описана выше. Данные прерывания можно игнорировать при соответствующих настройках процессора.

Маскируемость прерываний определяется флагом разрешения прерываний IF (Interrupt Flag), находящимся в регистре состояния EFLAGS процессора. При ***IF***=1 маскируемые прерывания разрешены, а при ***IF***=0 – запрещены. Для явного управления флагом ***IF*** в процессоре имеются чувствительные к уровню привилегий

инструкции разрешения маскируемых прерываний ***STI*** (SeT Interrupt flag) и запрета маскируемых прерываний ***CLI*** (Clear Interrupt flag).

Немаскируемое аппаратное прерывание происходит при появлении сигнала ***NMI*** (Non Maskable Interrupt) на входе процессора. Этот сигнал обрабатывается всегда вне зависимости от флага ***IF***. Немаскируемые прерывания сигнализируют о появлении критических событий для компьютера, например, о сбое по питанию. В ходе процедуры обслуживания немаскируемого прерывания процессор не реагирует на другие запросы немаскируемых и маскируемых прерываний до тех пор, пока не будет обработан. Если при обработке немаскируемого прерывания возникает новый сигнал ***NMI***, то он фиксируется и обрабатывается после завершения обработки текущего прерывания.

Исключения (exeprtions) делятся в процессоре Pentium на отказы (faults), ловушки (traps) и аварийные завершения (aborts). Данные прерывания похожи друг на друга – отличаются только адресом возврата.

Отказы (ошибки) соответствуют некорректным ситуациям, которые выявляются до выполнения инструкции, например, при обращении по адресу, находящемуся в отсутствующей в оперативной памяти странице (страничный отказ). После обработки исключения –отказа (загрузки страницы с жесткого диска), процессор повторяет выполнения команды, которую он не смог выполнить ранее.

Ловушки обрабатываются процессором после выполнения инструкции, например при возникновении переполнения. После обработки процессор выполняет инструкцию, следующую за той, которая вызвала исключение. Ловушки предназначены для отладки программ.

Аварийные завершения соответствуют ситуациям, когда невозможно точно определить команду, вызвавшую прерывание. Чаще всего это происходит во время серьезных отказов, связанных со сбоями в работе аппаратуры компьютера. Примером аварии служит исключение двойного нарушения, когда сама попытка обработки одного исключения вызывает другое исключение [1]. В этом случае обычно исполняемая программа завершает свою работу. Для обработки исключений в таблице прерываний отводятся номера 0-31 (таблица 4).

Таблица 4 – Типы исключений в защищённом режиме [1]

Номер вектора	Название	Описание	Тип	Код ошибки	Источник
0	#DE	Ошибка деления	Ошибка	Нет	Команды DIV и IDIV
1	#DB	Отладка	Ошибка или ловушка	Нет	Любая команда или команда INT 1
2	-	Прерывание NMI	Прерывание		Немаскируемое внешнее прерывание
3	#BP	Точка останова	Ловушка	Нет	Команда Int3
4	#OF	Переполнение	Ловушка	Нет	Команда INTO
5	#BR	Превышение предела	Ошибка	Нет	Команда BOUND
6	#UD	Недопустимая команда	Ошибка	Нет	Недопустимая команда или команда UD2
7	#NM	Устройство недоступно	Ошибка	Нет	Команды плавающей точки или команда WAIT/FWAIT
8	#DF	Двойная ошибка	Авария	Всегда 0	Любая команда
9	-	Зарезервировано компанией Intel. Не использовать!			
10	#TS	Недопустимый TSS	Ошибка	Да	Переключение задач или доступ к TSS
11	#NP	Сегмент отсутствует	Ошибка	Да	Загрузка сегментных регистров или доступ к сегментам
12	#SS	Ошибка сегмента стека	Ошибка	Да	Операции над стеком и загрузка в SS
13	#GP	Общая защита	Ошибка	Да	Любой доступ к памяти и прочие проверки защиты
14	#PF	Страничное нарушение	Ошибка	Да	Доступ к памяти
15	-	Зарезервировано компанией Intel. Не использовать!			
16	#MF	Ошибка плавающей точки в x87 FPU (ошибка математики)	Ошибка	Нет	Команда x87 FPU или команда WAIT/ FWAIT
17	#AC	Проверка выравнивания	Ошибка	Всегда 0	Обращение к памяти
18	#MC	Проверка оборудования	Авария	Нет	Наличие кодов и их содержимое зависит от модели
19	#XF	Исключение плавающей точки в SIMD	Ошибка	Нет	Команды SSE
20-31	-	Зарезервировано компанией Intel. Не использовать!			

Программные прерывания в процессоре Pentium происходят при выполнении инструкции INT с однобайтовым аргументом, в котором указывается вектор прерывания. Общая длина инструкции INT – два байта, исключение составляет инструкция INT 3, которая целиком помещается в один байт. Это удобно при отладке программ, когда инструкция INT заменяет первый байт любой команды, вызывая переход на процедуру отладки. Программные прерывания подобно ловушкам обрабатываются после выполнения соответствующей инструкции INT, а возврат происходит в следующую инструкцию. Программное прерывание может вызвать любую из 256 процедур обработки прерываний, указанных в таблице прерываний. Программные прерывания позволяют использовать обработчики прерываний наподобие обычных функций программы.

При одновременном поступлении запросов прерываний различных типов приоритет имеют немаскируемые прерывания. Приоритетность внутри маскируемых прерываний устанавливается контроллером прерываний (процессор не может этого сделать, так как для него все маскируемые запросы представлены одним сигналом INTR).

Таблица векторов прерываний в реальном режиме работы процессора всегда находится в адресном диапазоне оперативной памяти от 0x00000 по 0x003FF. В защищенном же режиме данная таблица называется IDT (Interrupt Descriptor Table) и располагается в любом месте ОЗУ. Ее начало (32-разрядный физический адрес) и размер (16 бит) хранятся в регистре системных адресов IDTR процессора. Каждый из 256 элементов дескрипторной таблицы прерываний представляет собой 8-байтный дескриптор. В таблице прерываний могут находиться только дескрипторы определенного типа – дескрипторы шлюзов прерываний, шлюзов ловушек и шлюзов задач.

Шлюзы прерываний и ловушек специально вводятся для вызова процедур обработки прерываний. Если для вызова процедуры обработки прерывания используется шлюз задач, то происходит смена процесса, а по завершении обработки – возврат к прерванному процессу. Шлюзы прерываний и ловушек не вызывают смены контекста задачи, следовательно, процедуры обработки

прерываний в этом случае вызываются быстрее, чем при использовании шлюза задачи.

В случаях, когда необходимо осуществить копирование данных из внешнего устройства в память или обратно часто используется режим прямого доступа к памяти DMA (Direct Memory Access). Для реализации данного способа обмена данными необходим аппаратный контроллер прямого доступа к памяти. Рассмотрим отличия данного способа обмена данными с внешними устройствами на примере копирования данных с жесткого диска.

Следующий алгоритм используется при обмене данными с внешним устройством на основе прерываний:

1 Контроллер считывает с диска один или несколько секторов в свой буфер.

2 Контроллер проверяет контрольную сумму для определения возможных искажений данных. При корректности данных контроллер инициирует прерывание.

4 Операционная система читает блок диска побайтно или пословно с контроллера и копирует в память. При этом фактически процессор считывает данные внешнего устройства в свои регистры, а потом выполняет копирование в память.

При использовании режима прямого доступа к памяти:

1 Центральный процессор программирует DMA-контроллер, записывая в его регистры управляющие команды для указания какие данные и куда следует переместить. Далее процессор передает управление системной шиной контроллеру прямого доступа к памяти.

2 DMA-контроллер начинает копирование данных, посылая дисковому контроллеру по шине запрос чтения и записи контроллеру оперативной памяти.

В целом, применение прямого доступа к памяти позволяет процессору переложить функции по формированию управляющих сигналов для копирования данных на контроллер DMA. Это разгружает процессор и позволяет ему в это время выполнять команды, которые находятся в кэш-памяти.

4.2 Принципы программного обеспечения системы ввода-вывода компьютера

Основными характеристиками программного обеспечения системы ввода-вывода являются:

- *независимость от устройств.* Что означает возможность написания программ, способных получать доступ к любому устройству ввода-вывода, без предварительного указания конкретного устройства;
- *единообразие именования.* Имя файла или устройства должно быть представлено просто текстовой строкой или целым числом и никоим образом не зависеть от физического устройства;
- *обработка ошибок.* Ошибки должны обрабатываться как можно ближе к аппаратуре. Если контроллер обнаружил ошибку, то он должен исправить её сам;
- *способ переноса данных*: синхронный (блокирующий) или асинхронный (управляемый прерываниями). Если мы работаем по прерываниям, т.е. асинхронный способ, то операционная система делает их для пользователя блокирующими – т.е. программа делает системный вызов и ожидает ответа;
- *буферизация.* Включает копирование данных в значительных размерах и увеличивает производительность операций ввода-вывода [20].

Важным является понятие выделенных устройств и устройств коллективного использования. К первым может иметь доступ только один пользователь в конкретный момент времени, а ко вторым – несколько пользователей.

Существуют три основных способа осуществления операций ввода-вывода:

- программный ввод-вывод;
- управляемый прерываниями ввод-вывод;
- ввод-вывод с использованием DMA.

Суть первого способа ввода-вывода рассмотрим на примере печати строки символов на принтере. В начале, процесс копирует данную строку в буфер. Далее принтер передается на определенный период времени процессу пользователя. После

этого процесс просит ОС распечатать строку. В свою очередь, операционная система сначала копирует буфер процесса в пространство ядра, и при готовности принтера для печати, копирует первый символ в регистр принтера. При этом операционная система смещает указатель на следующий символ. Данные операции повторяются, пока все символы не будут напечатаны. По окончании печати принтер становится доступным для выполнения следующих операций. Обобщенно данный процесс можно представить следующим образом (листинг 2):

```
copy_from_user(buffer,p,count);          /* p –буфер ядра */
for(i=0; i<count;i++){                   /* цикл символов */
      while (*printer_status_reg!=READY); /* цикл ожидания готовности */
      *printer_data_reg=p[i];            /* печать символа */
}
return_to_user();                        /*завершение системного вызова*/
```

Листинг 2 – Печать строки при помощи программного ввода-вывода

Особенностью данного способа обмена информацией является то, что после печати каждого символа процессор в каждой итерации опрашивает готовность устройства ввода-вывода, т.е. происходит активное ожидание. Программный ввод-вывод легко реализуется, не требует использования дополнительных аппаратных средств, но его основной недостаток – процессор используется все время во время ввода-вывода. При использовании «медленных» внешних устройств данное обстоятельство приводит к тому, что значительная часть времени работы процессора тратится впустую на активное ожидание. Такой подход приемлем только в примитивных встроенных системах. Данного недостатка можно избежать при обмене данными с внешними устройствами с использованием управляемого прерывания ввода-вывода. Ниже приведен пример печати строки на принтере для данного способа (листинг 3):

```
copy_from_user(buffer,p,count);
enable_interrupts();
while (*printer_status_reg!=READY);
*printer_data_reg=p[0];
scheduler(); (планировщик)
```

а

```
if (count==0) {
      unblock_user();
} else {
      *printer_data_reg=p[i];
      count=count-1;
      i=i+1;
}
return_from_interrupt();
```

б

Листинг 3 – Печать строки при помощи ввода-вывода, управляемого прерываниями: программа, выполняющая системный вызов для печати строки (а); процедура обработки прерываний (б)

Для печати строки пользовательская программа выполняет системный вызов, который копирует строку в буфер в ядра, и разрешается обработка прерываний. Далее копируется первый символ строки в регистр принтера при его готовности. Затем планировщик ОС блокирует текущий процесс, а печать строки осуществляет обработчик прерывания от принтера. В это время процессор может выполнять команды других процессов до прихода запроса на прерывание от принтера. При каждом сигнале готовности принтера вызывается обработчик прерывания, который выполняет копирование очередного символа. Текущий процесс разблокируется только тогда, когда закончится печать всей строки. Достоинством данного способа обмена является то, что процессор занимается передачей данных только в моменты готовности оборудования. Основной недостаток в том, что прерывания происходят при печати каждого символа. При этом обработка прерывания требует дополнительных временных затрат на его подготовку. Это приводит к тому, что доля служебных операций в процессе обмена данными с внешним устройством будет значительной.

Распространенным способом ввода-вывода является режим прямого доступа к памяти. В этом случае передачу символов в принтер осуществляет контроллер DMA, а не центральный процессор. Фактически данный способ отличается от предыдущего только тем, что всю работу выполняет DMA-контроллер. При этом

формируется только одно прерывание на передачу всего буфера символов, отправленного на печать.

Программное обеспечение системы ввода-вывода обычно состоит из четырех уровней, представленных на рисунке 35.

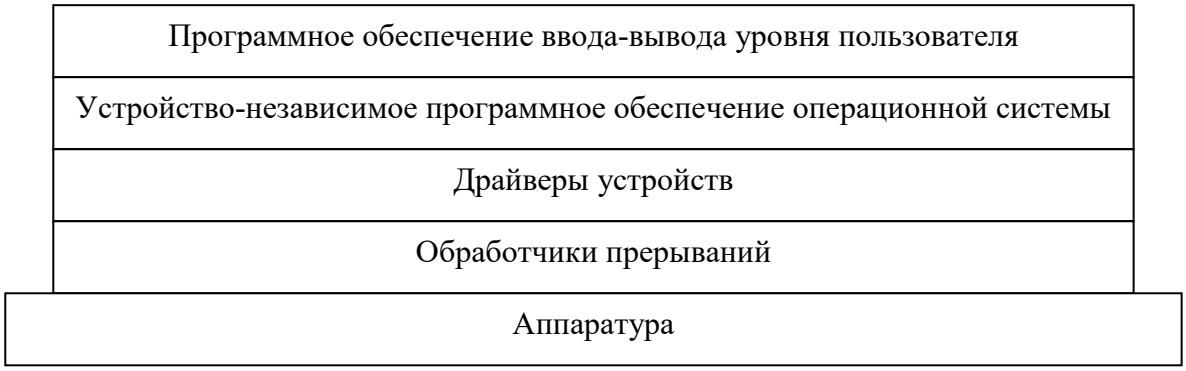


Рисунок 35 – Программные уровни системы ввода-вывода

В большинстве случаев обмен с внешними устройствами осуществляется на основе системы прерываний. При таком способе ввода-вывода, обычно в начале обмена данными драйвер устройства блокирует сам себя в ожидании запроса прерывания. При поступлении от аппаратного средства сигнала на прерывание, показывающее готовность к взаимодействию внешнего устройства, начинает работу обработчик прерывания. По окончании своей работы обработчик прерывания разблокирует драйвер, инициировавший процесс ввода-вывода.

В связи с тем, что каждое внешнее устройство является уникальным со своим набором регистров, протоколом управления, то для взаимодействия с ним производитель оборудования поставляет небольшую программу, называемую драйвером. Драйвер устройства должен выполняться на уровне ядра, чтобы иметь доступ к портам внешнего устройства. Через драйвер операционная система обращается к аппаратным средствам, так как показано на рисунке 36.

Операционная система обычно классифицирует драйверы на категории в соответствии с типами обслуживаемых устройств. К наиболее общим категориям относятся драйверы блочных и символьных устройств.

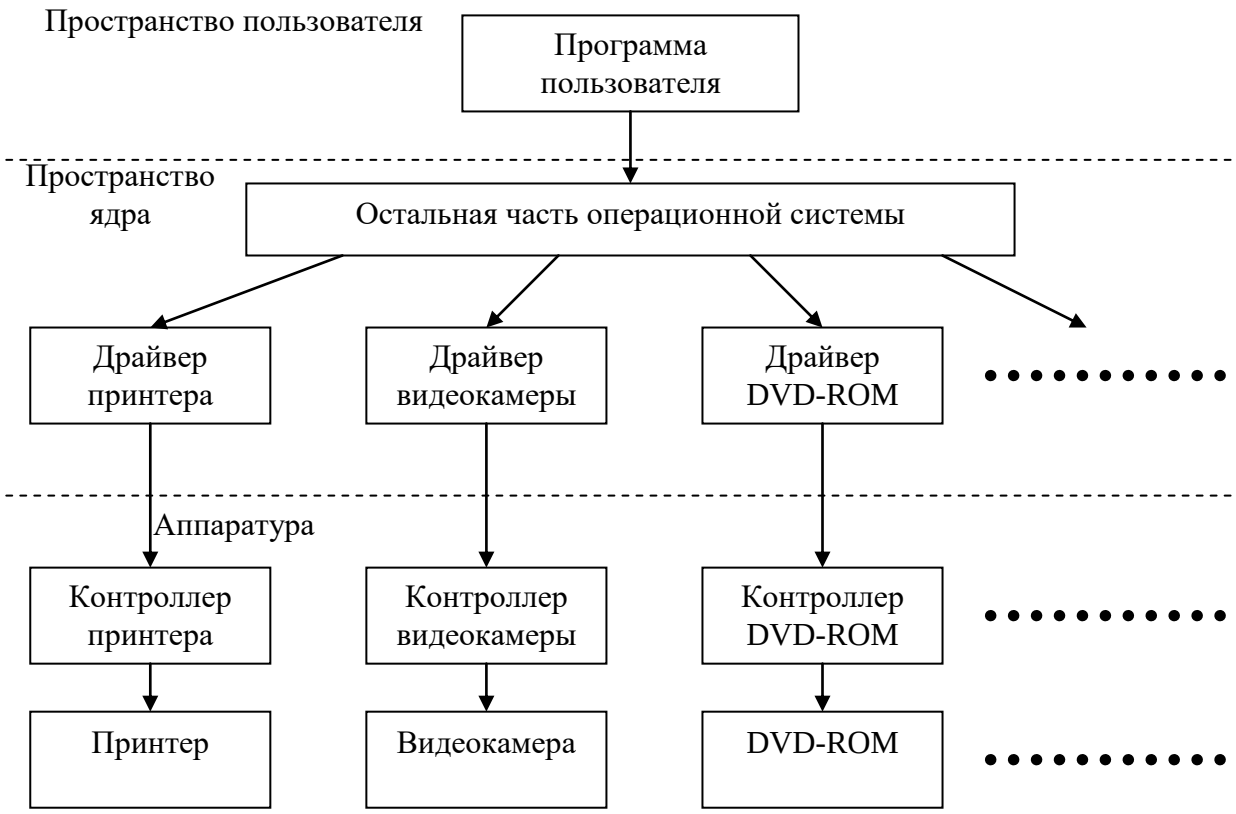


Рисунок 36 – Логическое расположение драйверов устройств

В большинстве операционных систем определен стандартный интерфейс, который должны поддерживать все блочные драйверы, и второй стандартный интерфейс, поддерживаемый всеми символьными драйверами. Данные интерфейсы представляют собой списки процедур, которые могут вызываться остальной частью ОС для обращения к драйверу. Например, к интерфейсу относятся процедуры чтения блока (блочного устройства) или записи символьной строки (для символьного устройства).

Основной функцией драйвера является обработка абстрактных, независимых от устройства, запросов ввода-вывода от программного обеспечения. При этом драйвер формирует управляющие команды для конкретного аппаратного устройства, учитывающие его структуру. Также драйверы могут при необходимости инициализировать, управлять энергопотреблением и регистрировать события внешнего устройства.

Управление аппаратным устройством подразумевает формирование последовательности команд, которые драйвер записывает в регистры внешнего устройства. В ответ устройство выполняет заданные операции, по окончании которых драйвер контролирует их безошибочность. При отсутствии ошибок данные передаются на уровень выше в независимое от устройства программное обеспечение, которое обычно реализует следующие функции:

- единообразный интерфейс для драйверов устройств;
- буферизация;
- сообщения об ошибках;
- захват и освобождения выделенных устройств;
- обеспечивает размер блока, независящий от устройства.

Основной задачей независимого от устройств ПО является предоставление единообразного интерфейса для ввода-вывода для программ уровня пользователя.

4.3 Основы файловых систем

Операционная система предоставляет удобный интерфейс пользователю для работы с данными, хранящимися на дисках. Для этого ОС подменяет физическую структуру хранящихся данных удобной для пользователя логической моделью. Логическая модель файловой системы материализуется в виде дерева каталогов, в символьных составных именах файлов, в командах работы с файлами. Базовым элементом данной модели является файл, который также характеризуется логической и физической структурой. Файл – это именованная область внешней памяти, в которую можно записывать или считывать из неё данные. Файлы хранятся в энергонезависимой памяти, например, на магнитных дисках. К исключению из этого правила можно отнести электронный диск, представляющий собой структуру в оперативной памяти и имитирующий файловую систему.

Основные цели использования файла:

- *долговременное и надежное хранение информации.* Долговременность достигается за счет использования запоминающих устройств, не зависящих от

питания, а высокая надежность достигается средствами защиты доступа к файлам и общей организацией программного кода ОС, при которой сбои аппаратуры чаще всего не разрушают информацию, хранящуюся в файлах;

- *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. Файл может быть создан одним пользователем, а затем использоваться совсем другим. При этом создатель файла или администратор могут определить права доступа к нему других пользователей[16].

Эти цели реализуются в операционной системе файловой системой. Файловая система – это часть операционной системы, включающая:

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

Файловая система играет роль промежуточного слоя, экранирующего все сложности физической организации долговременного хранилища данных, и создающего для программ более простую логическую модель этого хранилища, а также предоставляя им набор удобных в использовании команд для манипулирования файлами. Кроме того, 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.

Задачи, решаемые файловой системой, зависят от способа организации вычислительного процесса в целом. Самый простой тип – это файловая система в однопользовательских и однопрограммных операционных системах (например, MS-DOS). Основные функции в такой файловой системе нацелены на решение следующих задач:

- именование файлов;
- программный интерфейс для приложений;
- отображения логической модели файловой системы на физическую организацию хранилища данных;
- устойчивость файловой системы к сбоям питания, ошибкам аппаратных и программных средств.

Задачи файловой системы усложняются в операционных однопользовательских мультипрограммных системах. К перечисленным выше задачам добавляется новая задача совместного доступа к файлу из нескольких процессов. Файл в этом случае является разделяемым ресурсом, а значит, файловая система должна решать весь комплекс проблем, связанных с такими ресурсами. В частности, должны быть предусмотрены средства блокировки файла и его частей, исключение тупиков, согласование копий и т. п. В многопользовательских системах появляется еще одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя.

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых, как правило, входят обычные файлы, файлы-каталоги, специальные файлы, именованные конвейеры, отображаемые в память файлы и другие.

Обычные файлы, или просто файлы, содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ.

Каталоги – это особый тип файлов, которые содержат системную справочную информацию о наборе файлов, сгруппированных пользователями по какому-либо неформальному признаку.

Специальные файлы – это фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам.

Пользователи обращаются к файлам по символьным именам. Однако способности человеческой памяти ограничивают количество имен объектов, к которым пользователь может обращаться по имени. Иерархическая организация пространства имен позволяет значительно расширить эти границы. Именно поэтому большинство файловых систем имеет иерархическую структуру, в которой уровни создаются за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рисунок 37).

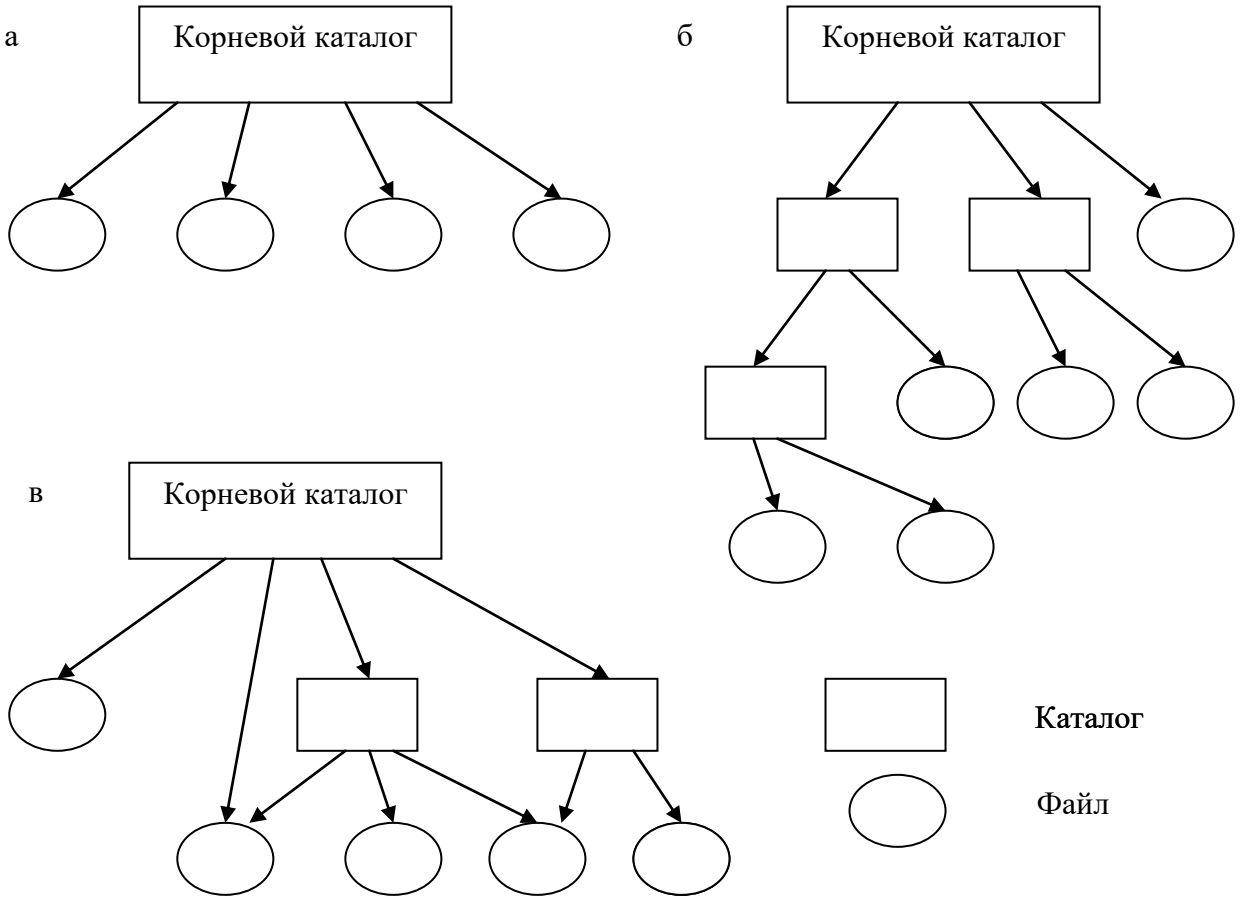


Рисунок 37 – Иерархия файловых систем

Каталоги образуют дерево, если файлу разрешено входить только в один каталог (рисунок 37, б), и сеть – если файл может входить сразу в несколько каталогов (рисунок 37, в). Например, в MS-DOS и Windows каталоги образуют

древовидную структуру, а в UNIX – сетевую. В древовидной структуре каждый файл является листом. Каталог самого верхнего уровня называется корневым каталогом, или корнем (root). Частным случаем иерархической структуры является одноуровневая организация, когда все файлы входят в один каталог (Рисунок 37, а).

Все типы файлов имеют символьные имена. В иерархически организованных файловых системах обычно используются три типа имен файлов: простые, составные и относительные. Простое, или короткое, символьное имя идентифицирует файл в пределах одного каталога. Полное имя представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до данного файла. Относительное имя файла определяется через понятие «текущий каталог».

При наличии нескольких устройств внешней памяти организация хранения файлов возможна двумя способами:

- на каждом устройстве размещается автономная файловая система (пример, MS-DOS, Windows);
- объединение в единую файловую систему с единым деревом каталогов, т.е. монтирование (UNIX).

Понятие «файл» включает не только хранимые им данные и имя, но и атрибуты. Атрибуты – это информация, описывающая свойства файла (тип файла, владелец файла, признак «только для чтения» и т.д.). Пользователь может получать доступ к атрибутам, используя средства, предоставленные для этих целей файловой системой. Обычно разрешается читать значения любых атрибутов, а изменять – только некоторые. Например, пользователь может изменить права доступа к файлу (при условии, что он обладает необходимыми для этого полномочиями), но изменять дату создания или текущий размер файла ему не разрешается.

Значения атрибутов файлов могут непосредственно содержаться в каталогах, как это сделано в файловой системе FAT. Другим вариантом является размещение атрибутов в специальных таблицах, когда в каталогах содержатся только ссылки на эти таблицы. Такой подход реализован, например, в файловой системе UFS. В этой файловой системе структура каталога очень простая. Запись о каждом файле

содержит короткое символьное имя файла и указатель на индексный дескриптор файла, так называется в UFS таблица, в которой сосредоточены значения атрибутов файла.

Файловые системы хранятся на дисках. Сектор 0 диска называется главной загрузочной записью MBR (Master Boot Record) и используется для загрузки компьютера. В конце MBR содержится таблица разделов, в которой хранятся начальные и конечные адреса каждого раздела. При загрузке компьютера BIOS считывает и исполняет MBR-запись, после чего определяется активный раздел и загрузчик в MBR-записи исполняет его. Программа, находящаяся в загрузочном блоке раздела, загружает операционную систему. Возможная структура файловой системы представлена на рисунке 38.

Суперблок содержит ключевые параметры файловой системы, включающие в себя количество блоков в файловой системе и другую административную информацию. I-узлы – это массив структур данных, содержащих информацию о файлах.

Важным моментом в реализации хранения файлов является учет соответствия блоков диска файлам. Для определения того, какой блок данных какому файлу принадлежит, в различных операционных системах применяются разные методы.

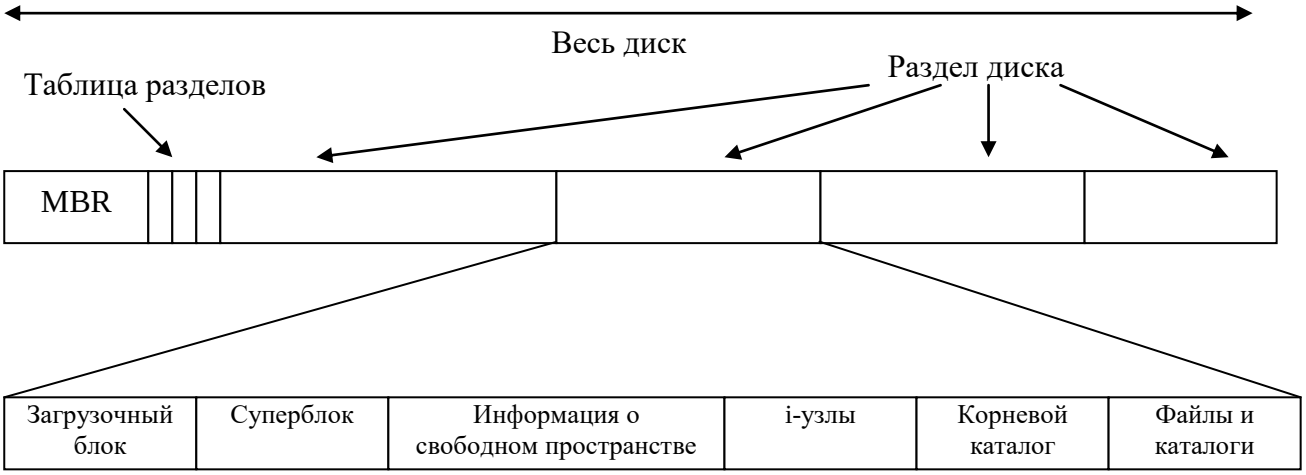


Рисунок 38 – Возможная структура файловой системы

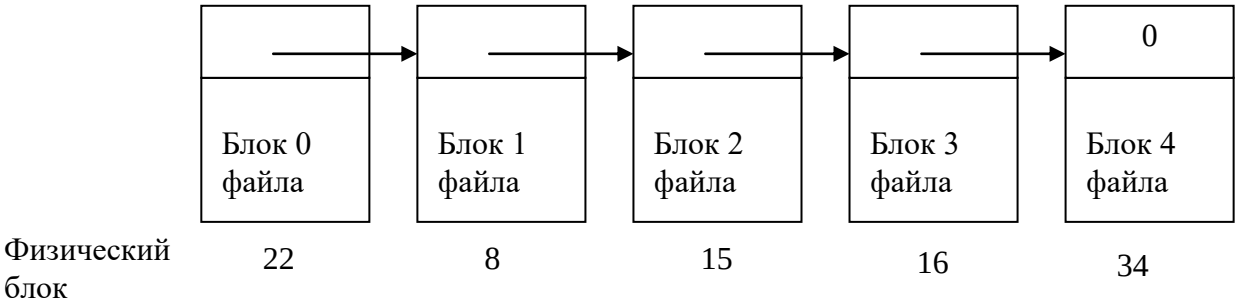


Рисунок 39 – Размещение файла в виде связного списка блоков диска

Реализация файлов возможна следующими способами:

1 *Непрерывные файлы.* Файлы представляют собой непрерывные наборы соседних блоков диска. Преимущества: простота реализации плюс высокая производительность, так как весь файл может быть прочитан с диска за одну операцию. Недостаток: в результате фрагментации необходимо будет знать конечный размер файла перед записью или постоянно производить дефрагментацию. Непрерывные файлы возможно применять на CD и DVD дисках.

2 *Связные списки.* Файл состоит из блоков диска, как показано на рисунке 39. Первое слово каждого блока используется как указатель на следующий блок. Недостатки: если мы хотим прочитать информацию в конце файла, то нам необходимо его читать сначала, что очень медленно. Кроме того теряется место на содержание указателей на следующий блок.

3 *Связный список при помощи таблицы в памяти.* В отличии от предыдущего способа все указатели на следующие блоки хранятся в отдельной таблице. Недостаток: вся таблица, которая называется FAT-таблицей, должна находиться в оперативной памяти и будет занимать её значительную часть. Так для 20-гигабайтного диска с блоками размером 1 Кбайт потребовалась бы таблица из 20 миллионов записей, каждая из которых не менее 4 байт. А это 80 Мбайт постоянно занятой оперативной памяти.

4 *I-узлы.* В этом случае с каждым файлом связывается структура данных (i-узел – index-узел), содержащая атрибуты файлов и адреса блоков файла. Перед

работой с файлом i-узел читает в память все адреса блоков. Преимущество: для реализации требуется небольшой объем памяти.

4.4 Файловая система FAT

Файловая система FAT (File Allocation Table – файловая таблица распределения) является одной из простейших систем. Основная концепция файловой системы FAT заключается в том, что каждому файлу и каталогу выделяется структура данных, называемая записью каталога. В этой структуре хранится имя файла, его размер, начальный адрес содержимого файла и другие метаданные. Содержимое файлов и каталогов хранится в блоках данных, называемых кластерами. Если файлу или каталогу выделяется более одного кластера, остальные кластеры находятся при помощи структуры данных, называемой FAT. Существуют 3 версии FAT: FAT12, FAT16 и FAT32. Они отличаются между собой размером записей в структуре FAT. На рисунке 40 показана общая схема между структурами данных [8].

Файловая система FAT делится на три физические области:

- зарезервированная область, в которой хранятся данные из категории файловой системы. Размер её определяется в загрузочном секторе. В FAT12 и FAT16 занимает всего 1 сектор;
- область FAT – содержит основные и резервные структуры FAT. Она начинается в секторе, следующем за зарезервированной областью, а её размер определяется количеством и размером структур FAT;
- область данных содержит кластеры, выделяемые для хранения файлов и содержимого каталогов (рисунок 41).

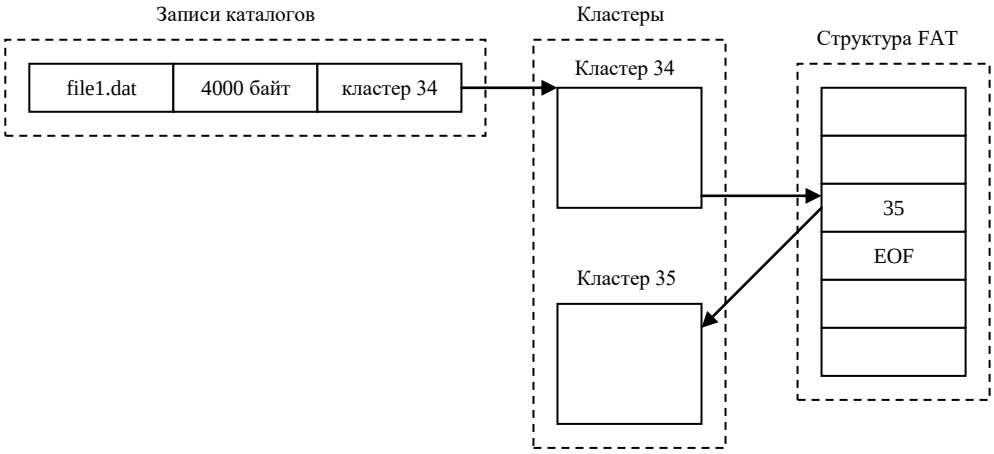


Рисунок 40 – Отношения между записями каталогов, кластерами и FAT

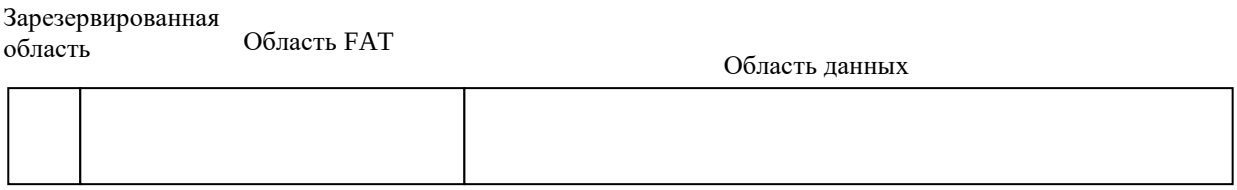


Рисунок 41 – Физическая структура файловой системы FAT

Для того, чтобы открыть файл операционная система должна прочитать соответствующую запись каталога. Первоначальная каталоговая запись системы FAT представлена на рисунке 42.

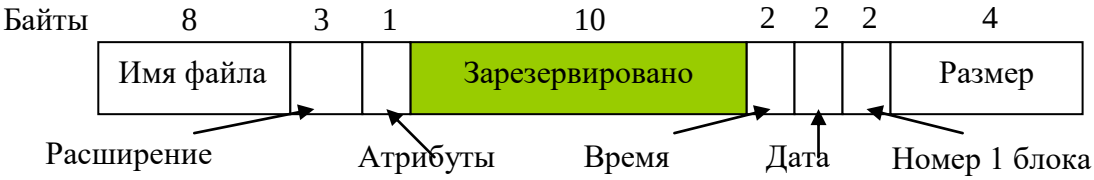


Рисунок 42 – Формат каталоговой записи в системе FAT

FAT-12, 16 и 32 различаются размерами минимальных блоков (кластеров), общим максимальным объемом диска и разрядностью указателей на эти блоки (12, 16, 28 разряда). FAT-12 применялась на гибких дисках. Размер дискового раздела мог составлять 2 Мб, а размер блока 512 байт, 1 Кб, 2 Кб, 4 Кб. FAT-16 вы можете использовать и сейчас, например отформатировав флеш-носитель небольшого размера. Кластеры здесь размером 8, 16 или 32 Кб. Максимальный размер дискового

раздела (логический диск) – 2 Гб, максимальный размер диска – 8 Гб. Таблица FAT занимает в памяти 128Кб. В FAT32 размер разделов ограничен 2Тб (2048 Гб). Размеры кластеров остались прежними. FAT32 широко используется и по сей день. При этом изменился формат каталоговой записи, который представлен на рисунке 43.

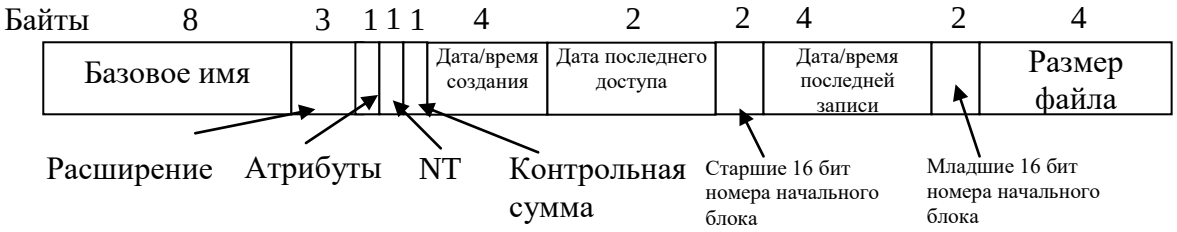


Рисунок 43 – Формат каталоговой записи в системе FAT32

Если у файла длинное имя, то оно хранится в одной или нескольких каталоговых записях, предшествующих описателю файла (рисунок 40). Каждая такая запись содержит до 13 символов формата Unicode. Элементы имени хранятся в обратном порядке, начинаясь сразу перед описателем файла в формате MS-DOS и последующими фрагментами перед ним. Формат каждого фрагмента имени представлен на рисунке 44.

Операционная система отличает стандартные каталоговые записи от записей с фрагментом длинного имени файла по полю Attributes (атрибуты). Для фрагмента длинного имени это поле содержит значение 0x0F, что соответствует невозможной комбинации атрибутов для описателя файла в MS-DOS. Старые программы, написанные для работы в MS-DOS, читая каталог, просто игнорируют такие описатели как неверные.

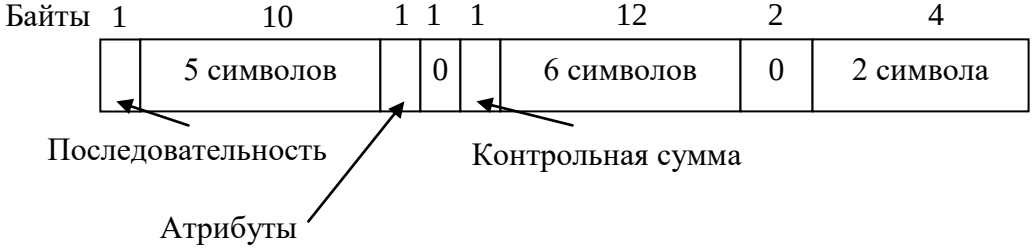


Рисунок 44 – Формат каталоговой записи с фрагментом длинного имени файла

Реализация файловой системы FAT-32 концептуально близка к реализации файловой системы FAT-16. Однако вместо массива из 65 536 элементов в ней используется столько, сколько нужно, чтобы покрыть весь раздел диска. Если диск содержит миллион блоков, то и таблица будет состоять из миллиона элементов. Для экономии памяти система Windows 98 не хранит их все сразу в памяти, а использует окно, накладываемое на таблицу.

4.5 Файловая система NTFS

Основными целями при проектировании NTFS (New Technology File System – файловая система новой технологии) были надежность, безопасность и поддержка носителей информации большой емкости. Основные особенности файловой системы NTFS следующие:

- *способность восстановления данных.* Файловая система восстанавливается при отказе системы и сбоев дисков. Это достигнуто посредством использования транзакций, позволяющих осуществлять журналирование файловых операций;
- *безопасность.* Файловая система поддерживает объектную модель безопасности и рассматривает все тома, каталоги, файлы как самостоятельные объекты. NTFS обеспечивает безопасность на уровне файлов. Это означает, что право доступа к файлам зависит от учетной записи пользователя, и тех групп, к которым он принадлежит;
- *расширенная функциональность.* NTFS проектировалась с учетом возможного расширения. В ней реализованы такие возможности, как эмуляция других операционных систем, параллельная обработка потоков данных и создание файловых атрибутов определенных пользователем;
- *поддержка POSIX (Portable Operating System for computing environments).* Международный стандарт машинно-независимого интерфейса вычислительной среды. В нем основное внимание уделяется взаимодействию

прикладных программ с операционной системой. Прикладная программа, написанная в соответствии с данным стандартом, легко переносится из одной операционной системы в другую;

- *эффективная поддержка больших дисков и файлов.* Максимальный размер тома NTFS составляет 2^{64} байт = 1 Экзобайт = 16000 млр. Гб. Максимальный размер файла составляет 2^{32} кластера = 2^{64} байт. Размер кластера может меняться от 512 байт до 64 Кбайт. NTFS поддерживает длинные имена файлов, набор символов Unicode и имена 8.3. Количество файлов в корневом и не корневом каталоге не ограниченно [8].

NTFS не обладает жестко заданной структурой. Вся файловая система считается областью данных, и любой сектор может быть выделен файлу. Единственное фиксированное требование – это первые сектора тома содержат загрузочный сектор и загрузочный код.

«Сердцем» NTFS является главная файловая таблица MFT (Master File Table – общая таблица файлов), содержащая информацию обо всех файлах и каталогах. Каждый файл или каталог представлен как минимум одной записью таблицы, причём записи сами по себе очень просты. Их размер составляет 1 Кбайт, но только первые 42 байта имеют определенное предназначение. В остальных байтах хранятся атрибуты – небольшие структуры данных, выполняющие строго специализированную функцию. Например, один атрибут используется для хранения имени файла, а другой – для хранения его содержимого. На рисунке 45 показана основная структура записи MFT с заголовком и тремя атрибутами.

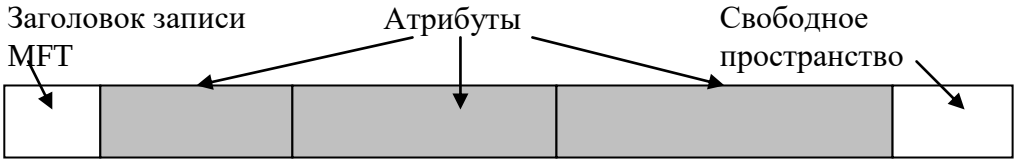


Рисунок 45 – Структура записи MFT

Количество атрибутов зависит от версии NTFS и характеристик описываемого объекта. Атрибут – это объект, содержащий данные определенного типа.

Существуют атрибуты для имени файла, даты, времени и даже для содержимого файлов. В этом проявляется одно из отличий NTFS от других файловых систем. Как правило, файловые системы читают и записывают содержимое файлов, а NTFS читает и записывает атрибуты, одна из разновидностей которых передаёт содержимое файлов. В таблице 5 перечислены некоторые стандартные типы атрибутов и соответствующие им идентификаторы. Не все типы атрибутов и идентификаторы существуют для каждого файла.

Таблица 5 – Некоторые стандартные типы атрибутов в записях MFT

Идентификатор типа	**Имя**	**Описание**
16	$STANDARD_INFORMATION	Общая информация (флаги; время создания, последнего обращения и модификации; владелец и идентификатор системы безопасности)
32	$ATTRIBUTE_LIST	Список других атрибутов файла
48	$FILE_NAME	Имя файла в Unicode; время создания, последнего обращения и модификации
64	$VOLUME_VERSION	Информация о томе. Существует только в версии 1.2
80	$SECURITY_DESCRIPTOR	Время обращения и свойства безопасности файла
96	$VOLUME_NAME	Имя тома
112	$VOLUME_INFORMATION	Версия файловой системы и другие флаги
128	$DATA	Содержимое файла
144	$INDEX_ROOT	Корневой узел индексного дерева

Атрибут состоит из заголовка и содержимого. Заголовок определяет тип атрибута, его размер и имя, содержит флаги, указывающие на сжатие или шифрование.

Содержимое атрибутов имеет произвольные форматы и размеры. Естественно, что неудобно сохранять такое количество данных в 1Кбайтных записях MFT. Для решения этой проблемы в NTFS предусмотрена возможность хранения содержимого атрибутов в двух местах: резидентные атрибуты хранятся в MFT записях с заголовками, нерезидентные атрибуты хранятся во внешних кластерах файловой системы. Что представлено на рисунке 46.

Теоретически файл может содержать до 65536 атрибутов (из-за 16-разрядных идентификаторов), поэтому для хранения всех заголовков атрибутов одной записи MFT может быть недостаточно. Поэтому создается базовая MFT-запись и ссылается на другие MFT-записи. Чтобы уменьшить объем места, занимаемого файлом, NTFS может сохранять значения некоторых нерезидентных атрибутов в разреженном формате, т.е. заполненные нулями кластеры не записываются на диск. NTFS позволяет хранить атрибуты в сжатом виде, а также применять шифрование атрибутов.

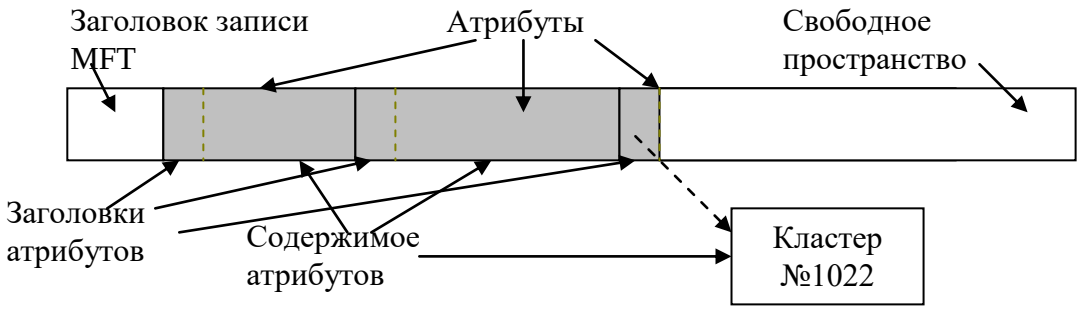


Рисунок 46 – Структура записи MFT с заголовками и содержимым атрибутов

Таким образом, структура раздела файловой системы NTFS имеет вид, представленный на рисунке 47. Первые 12 % диска отводятся под так называемую MFT зону – пространство, в которое растут MFT записи. Запись каких-либо данных в эту область невозможна. MFT-зона всегда держится пустой – это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 88% диска представляют собой обычное пространство для хранения файлов.

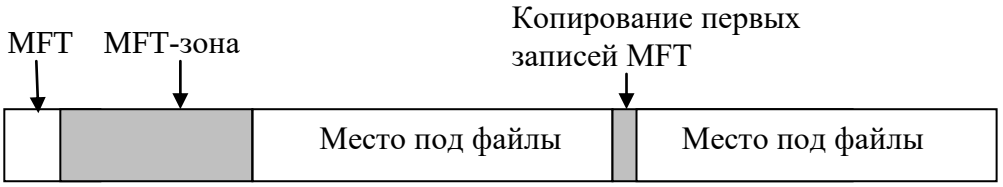


Рисунок 47 – Структура раздела NTFS

Для упрощения поиска в NTFS используются индексные структуры данных, в которых атрибуты сортируются в виде B-деревьев. Деревом называется

совокупность структур данных, называемых узлами; узлы связываются между собой, начиная с корневого узла. Пример структуры данных в виде дерева приведен на рисунке 48, а.

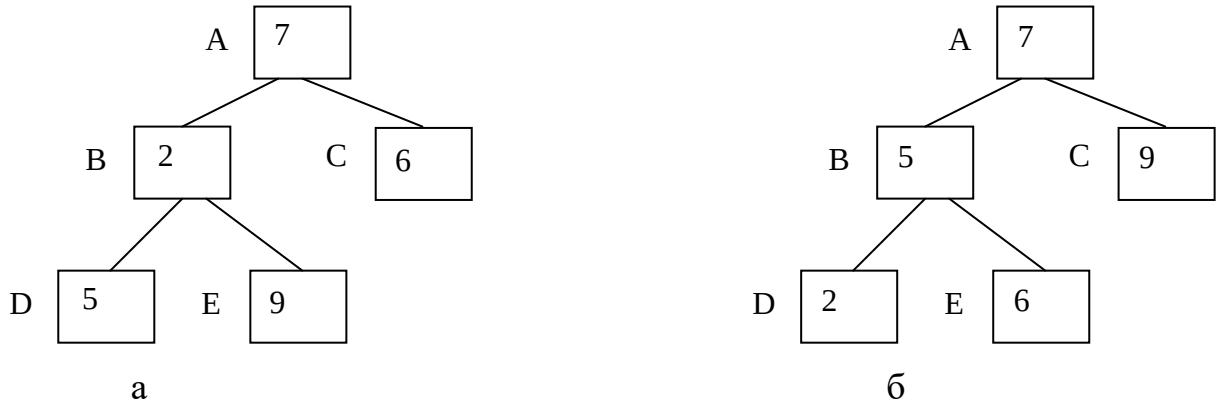


Рисунок 48 – а) дерево с пятью узлами; б) то же дерево после сортировки узлов

Родительским называется узел, от которого идут связи к другим узлам, а дочерним – к кому направлены связи. Использование структуры данных в виде B-дерева облегчает поиска элемента, в отличие от последовательного перебора значений по списку. В случае использования B-дерева, в каждой операции сравнения осуществляется практически деление списка на два подмножества, и переход по линиям связи направо или налево. В NTFS используется схожая структура индексации, позволяющая ускорить поиск элементов, но при этом усложняется процедура добавления и удаления файла в дерево. То же касается и восстановления данных.

Файловая система NTFS достаточно сложна и полное описание функционирования системы и её возможностей займёт отдельную книгу. Дополнительную информацию об NTFS можно почерпнуть в следующих публикациях [8, 13, 14].

4.6 Контрольные вопросы

1. Назовите отличия блочных и символьных внешних устройств.

2. Назовите достоинства и недостатки отображения адресного пространства внешних устройств на адресное пространство памяти.

3. В чем недостатки программно-управляемого обмена данными с внешними устройствами?

4. Опишите принцип работы ввода-вывода с использованием режима прямого доступа к памяти.

5. Определите понятия прерывание, обработчик прерывания.

6. Определите достоинства и недостатки обмена данными с внешними устройствами на основе системы прерываний?

7. Назначение таблицы векторов прерываний.

8. Дайте определение драйвера внешнего устройства. Назовите его назначение.

9. Назовите основные функции файловой системы.

10. Чем отличаются функции многопользовательских многопрограммных файловых систем от однопользовательской однопрограммной?

11. Чем отличается иерархия файловых систем UNIX и Windows?

12. Что такое MBR и зачем она нужна?

13. В чём недостаток реализации файлов связными списками?

14. Объясните назначение структуры FAT и её нахождение в памяти?

15. Как хранятся длинные имена файлов в системе FAT-32?

16. Перечислите основные преимущества системы NTFS.

17. Что такое MFT-таблица и MFT-запись?

5 Безопасность операционных систем

5.1 Основы безопасности

Основными задачами системы зашиты компьютерной системы являются:

- конфиденциальность;
- целостность;
- доступность системы.

Первая задача, обеспечение конфиденциальности данных, заключается в том, что они должны оставаться секретными для лиц, которые не имею права доступа. Вторая задача - обеспечения целостности данных, означает то, что неавторизованные пользователи не должны иметь возможности модифицировать данные без разрешения владельца. Третья задача, доступность системы, означает, что система с высокой вероятностью сможет обслужить запросы клиентов. Это достигается за счет снижения простоев в работе компьютерной системы за счет повышения её надежности и уровня защищенности.

Каждой задаче соответствует свой класс угроз:

- демонстрация данных;
- порча и подделка данных;
- отказ в обслуживании.

Источником угроз информационной безопасности чаще всего являются злоумышленники, которых можно разделить на два класса: активных и пассивных. Пассивные злоумышленники просто пытаются прочитать данные, к которым у них не прав доступа. Активные же пытаются незаконно модифицировать данные. Наиболее распространёнными категориями злоумышленников являются:

- случайные любопытные пользователи, не применяющие специальных технических средств;
- злоумышленники, совершающие попытки личного обогащения;
- злоумышленники, занимающиеся коммерческим и военным шпионажем.

Помимо преднамеренных угроз ИБ со стороны злоумышленников существует опасность случайной потери данных вследствие следующих причин:

- форс-мажор: пожар, землетрясение и т.п.;
- аппаратные и программные ошибки: сбои центрального процессора, ошибки при передаче данных по каналам связи, ошибки в программном обеспечении;
- человеческий фактор: неправильный ввод данных, удаление и т.п.

Наиболее эффективным способом защиты от нарушения конфиденциальности в многопользовательских операционных системах является использование криптографического преобразования данных. Основной задачей криптографии является шифровании данных. Схематично процесс шифрования представлен на рисунке 49. Открытый текст ***P*** шифруется при помощи функции ***E*** и ключа шифрования $\boldsymbol{K_E}$. В результате получается зашифрованный текст, который может быть дешифрирован функцией ***D***.

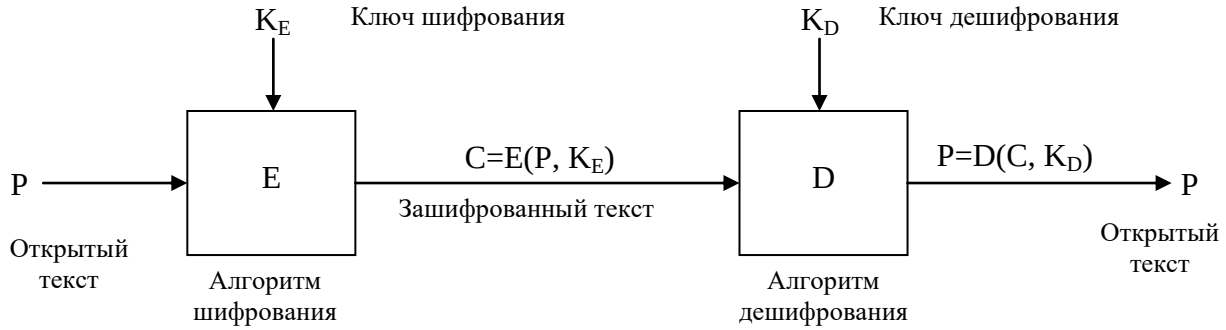


Рисунок 49 – Схема шифрования

Криптостойкость системы шифрования зависит от таких параметров, как выбранный алгоритм шифрования (ключей), надежность хранения ключевой информации. Существуют алгоритмы шифрования с секретными (симметричные) и открытыми ключами (ассиметричные). Алгоритмы с секретным ключом эффективны с точки зрения производительности и используются в случаях наличия у получателя и отправителя одинакового ключа. Важнейшей задачей для подобных систем является безопасное распространение ключа между участниками обмена данными. Наиболее известными алгоритмами симметричного шифрования являются

DES, AES. В алгоритмах с открытым ключом для шифрования и дешифрования используются различные ключи: открытый и закрытый. Открытый ключ используется только для дешифрования и передается всем. Закрытый ключ должен надежно хранится у отправителя и использоваться только для шифрования информации. В основном ассиметричные алгоритмы шифрования используются для формирования электронной подписи, взаимной аутентификации. К данному классу алгоритмов шифрования относится RSA. Особенностями ассиметричных алгоритмов шифрования являются высокая степень криптостойкости и значительная вычислительная сложность.

Угрозы безопасности операционной системе можно классифицировать по различным признакам [21].

1. По цели атаки:

- несанкционированное чтение информации;
- несанкционированное изменение информации;
- несанкционированное уничтожение информации;
- полное или частичное разрушение операционной системы.

2. По принципу воздействия на операционную систему:

- использование известных (легальных) каналов получения информации, например, угроза несанкционированного чтения файла, доступ пользователей к которому определен некорректно, т. е. разрешен доступ пользователю, которому согласно политике безопасности доступ должен быть запрещен;
- использование скрытых каналов получения информации, например, угроза использования злоумышленником недокументированных возможностей операционной системы;
- создание новых каналов получения информации с помощью программных закладок.

3. По типу используемой злоумышленником уязвимости защиты:

- неадекватная политика безопасности, в том числе и ошибки администратора системы;

- ошибки и недокументированные возможности программного обеспечения операционной системы, в том числе и так называемые люки — случайно или преднамеренно встроенные в систему «служебные входы», позволяющие обходить систему защиты;
- ранее внедренная программная закладка.

4. По характеру воздействия на операционную систему:

- активное воздействие — несанкционированные действия злоумышленника в системе;
- пассивное воздействие — несанкционированное наблюдение злоумышленника за процессами, происходящими в системе.

5.2 Аутентификация пользователей

Одним из важнейших элементов системы защиты операционной системы от несанкционированного доступа является подсистема аутентификации. Аутентификация (authentication) позволяет осуществить доступ к вычислительной системе только для зарегистрированных пользователей. Термин «аутентификация» в переводе с латинского означает «установление подлинности». Аутентификацию следует отличать от идентификации. Идентификаторы пользователей используются в системе с теми же целями, что и идентификаторы любых других объектов, файлов, процессов, структур данных, но они не связаны непосредственно с обеспечением безопасности. Идентификация заключается в сообщении пользователем системе своего идентификатора, в то время как аутентификация – это процедура доказательства пользователем того, что он есть тот, за кого себя выдает, в частности, доказательство того, что именно ему принадлежит введенный им идентификатор.

Возможны три способа аутентификации:

- аутентификация с использованием паролей;
- аутентификация с использованием физического объекта;

- аутентификация с использованием биометрических данных.

Наиболее широко применяется аутентификация с использованием паролей, которая легко реализуется. Обычно эту защиту обходят простым перебором комбинаций паролей и имен пользователей. Для уменьшения вероятности нахождения пароля злоумышленником за заданный интервал времени требуется, чтобы он представлял собой случайную последовательность символов большого размера. Поэтому в качестве требований к паролям обычно выступают следующие:

- пароль должен содержать как минимум семь символов;
- пароль должен содержать как строчные, так и прописные символы;
- пароль должен содержать как минимум одну цифру или специальный символ;
- пароль не должен представлять собой слово, содержащееся в словаре, собственное имя и т.д.

При использовании многоразового пароля особенно при сетевом доступе есть большой недостаток – его может перехватить злоумышленник и использовать в своих целях. Более надежными оказываются схемы, использующие одноразовые пароли, которые, как правило, рассчитаны на аутентификацию удалёнными пользователями. Генерация одноразовых паролей может выполняться либо программно, либо аппаратно. Независимо от того, какую реализацию системы аутентификации на основе одноразовых паролей выбирает пользователь, он, как и в системах аутентификации с использованием многоразовых паролей, сообщает системе свой идентификатор, однако вместо того, чтобы вводить каждый раз один и тот же пароль, указывает последовательность цифр, сообщаемую ему аппаратным или программным ключом. Через определенный небольшой период времени генерируется другая последовательность – новый пароль. Подобный способ аутентификации часто применяется в web-сервисах, когда одноразовый пароль передается в виде SMS-сообщения на телефон.

Еще один вариант реализации идеи паролей заключается в том, что для каждого нового пользователя создается длинный список вопросов и ответов, который хранится на сервере в надежном виде (например, в зашифрованном виде).

Вопросы должны выбираться так, чтобы пользователю не нужно было их записывать. При регистрации сервер задает один из этих вопросов, выбирая его из списка случайным образом, и проверяет ответ. Однако, чтобы такая схема могла работать, потребуется большое количество пар вопросов и ответов.

Другой вариант называется «оклик-отзыв». Он работает следующим образом. Пользователь выбирает алгоритм, идентифицирующий его как пользователя, например x^2. Когда пользователь входит в систему, сервер посылает ему некое случайное число, например 7. В ответ пользователь посылает серверу 49. Алгоритм может отличаться утром и вечером, в различные дни недели и т. д.

Второй способ аутентификации заключается в проверке наличия некоторого физического объекта у пользователя. К таким объектам относятся пластиковые карты (например, для банкомата), смарт-карты, которые содержат в себе примитивную операционную систему, сотовые телефоны. Отличие смарт-карты от обычной пластиковой в том, что у неё есть центральный процессор.

Со смарт-картами могут применяться различные схемы аутентификации. Простой протокол «оклик-отзыв» работает следующим образом. Сервер посылает 512-разрядное случайное число смарт-карте, которая добавляет к нему 512-разрядный пароль, хранящийся в электрически стираемом программируемом ПЗУ. Затем сумма возводится в квадрат, и средние 512 бит посылаются обратно на сервер, которому известен пароль пользователя, поэтому сервер может произвести те же операции и проверить правильность результата. Эта последовательность операций показана на рисунке 50.

Если даже злоумышленник видит оба сообщения, он не может определить по ним пароль. Сохранять эти сообщения также нет смысла для взломщика, так как в следующий раз сервер пошлет пользователю другое 512-разрядное случайное число. Конечно, вместо возведения в квадрат может применяться (и, как правило, применяется) более сложный алгоритм.

Недостаток любого фиксированного криптографического протокола состоит в том, что со временем он может быть взломан, в результате чего смарт-карта станет бесполезной. Избежать этого можно, если хранить в памяти карты не сам

криптографический протокол, а интерпретатор Java. При этом настоящий криптографический протокол будет загружаться в карту в виде двоичной программы Java и исполняться на ней. Таким образом, как только один протокол будет взломан, можно мгновенно перейти на использование другого протокола. Недостаток такого подхода заключается в том, что и без того не отличающаяся высокой производительностью смарт-карта будет работать еще медленнее, однако с развитием технологий этот метод приобретает все большую гибкость.

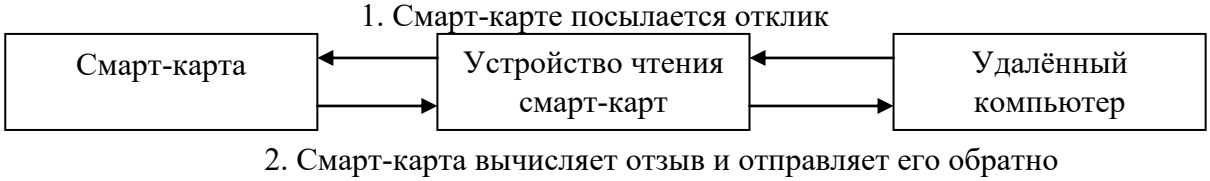


Рисунок 50 – Использование смарт-карты для аутентификации

Третий способ аутентификации основан на измерении физических характеристик пользователя, которые трудно подделать, таких как клавиатурный почерк, длина пальцев, отпечатки пальцев, рисунок сетчатки глаза, голос и т.д. Некоторые методы описаны в работе [2]. Достоинством данного способа является то, что аутентифицирующая информация привязана к пользователю и неотчуждаема от него. Но данный вариант характеризуется высокой стоимостью необходимых аппаратных средств и наличием погрешности распознавания биометрических признаков. Например, царапины на пальцах, могут привести к тому, что система не сможет распознать отпечаток.

Таким образом, образом в зависимости от ценности информации, обрабатываемой в вычислительной системе, используются различные способы аутентификации. В системах, в которых не требуется высокий уровень защищенности, достаточно применение парольной системы аутентификации. В случаях, когда обрабатывается ценная информация, лучше использовать методы аутентификации на основе наличия физических объектов или использования биометрических признаков. Также для последних систем используются многофакторные методы аутентификации, в частности, двухфакторные, использующие, например, пароль и смарт-карту.

5.3 Атаки на операционные системы

Операционная система может подвергнуться различным типам атак [21]: сканированию файловой системы, подбору пароля, краже ключевой информации, сборке мусора, превышению полномочий, программным закладкам и жадным к ресурсам программам.

Зарегистрировавшись на компьютере, взломщик может выполнить атаку и нанести ущерб. Если на компьютере установлена надежная система безопасности, возможно, взломщик сможет навредить только тому пользователю, чей пароль он уже взломал. Но часто начальная авторизация в системе может использоваться в качестве ступеньки для последующего взлома других учетных записей.

Троянские кони.

Это вредоносная программа, используемая злоумышленником для сбора информации, её разрушения или модификации, нарушения работоспособности компьютера или использования его ресурсов в неблаговидных целях. В последние годы часто троянские программы используются для осуществления несанкционированных операций через системы дистанционного банковского обслуживания. Особенно это актуально в случаях, когда сотрудники не соблюдают меры безопасности и используют пароли по умолчанию, забывают в компьютерах аппаратные ключи с секретной информацией.

Часто для заражения компьютера троянской программой пользователь должен запустить его сам. Поэтому нужно быть осторожным при закачке бесплатных данных и программ из сомнительных источников.

Фальшивая программа регистрации.

Данный вид атаки заключается в том, что пользователь вводит пароль в окне вредоносной программы взломщика. Для того, чтобы взломщик не смог подменить окно смены пользователя, в Windows для её вызова используется защищенная от перехвата комбинация клавиш CTRL+ALT+DEL. Однако и для этой комбинации есть способы взлома, но в этом случае злоумышленнику необходимо внедрить DLL в процесс Winlogon, что является относительно сложным.

Логические бомбы.

Логическая бомба – это написанная одним из сотрудников программа, тайно установленная в операционную систему, которая при внезапно увольнении сотрудника через некоторое время начинает действовать: например, форматировать жесткий диск, удалять файлы в случайном порядке или шифровать важные файлы.

Переполнение буфера.

Практически все операционные системы написаны на языке С, программы на котором компилируются гораздо эффективнее, чем на других языках. Однако ни один из компиляторов С не проверяет границы массива – за этим должен следить сам программист. Пример программного кода, в котором присутствует подобная ошибка, представлен в листинге 7.

```
int i;
BYTE buf[10000];
i=20000;
buf[i]=0;
```

Листинг 4 – Пример обращение к памяти за границей массива

В результате выполнения программы изменяется байт вне участка памяти, выделенного под массив. Это свойство языка С позволяет произвести атаку следующего типа. На рисунке 51, а показана работающая программа со своими переменными в стеке. В некоторый момент она вызывает процедуру ***A*** (рисунок 51, б). В стек помещается адрес возврата и управление передается процедуре ***A***. Для определенных целей в процедуре зарезервирован буфер ***B*** (например, для имени файла), однако пользователь вводит данные, размер которых больше объема буфера. И если процедура не проверяет размер вводимых данных, то адрес возврата в программу может затереться. Если предположить, что взломщик знает размер стека, то он может сделать буфер не из случайных байтов, а из команд, а адрес возврата точно подгадать так, как показано на рисунке 51, в. В результате злоумышленник сможет выполнить свой программный код при отсутствии на это прав. Защита от

атак подобного рода заключается в проверке длины всех поставляемых пользователем строк перед копированием в буфер.

В связи с распространением Интернет и локальных сетей всё большую угрозу представляют атаки операционной системы с удалённых компьютеров. Во многих случаях атака состоит из того, что по сети передаётся вредоносная компьютерная программа, при выполнении которой атакуемой машине наносится ущерб.

Особое внимание здесь стоит уделять вирусам. Вирус – это программа, которая может размножаться, присоединяя свой код к другой программе. Возможные сценарии нанесения ущерба вирусом:

- безвредные действия, вроде вывода сообщений на экран, изображений, воспроизведения звука и т.п.;
- порча информации, возможно шифрования файлов с целью шантажа;
- атака с целью отказа в обслуживании. Использование вирусом всех ресурсов компьютера, например, процессорного времени, заполнения жесткого диска;
- вывод из строя аппаратного обеспечения компьютера.

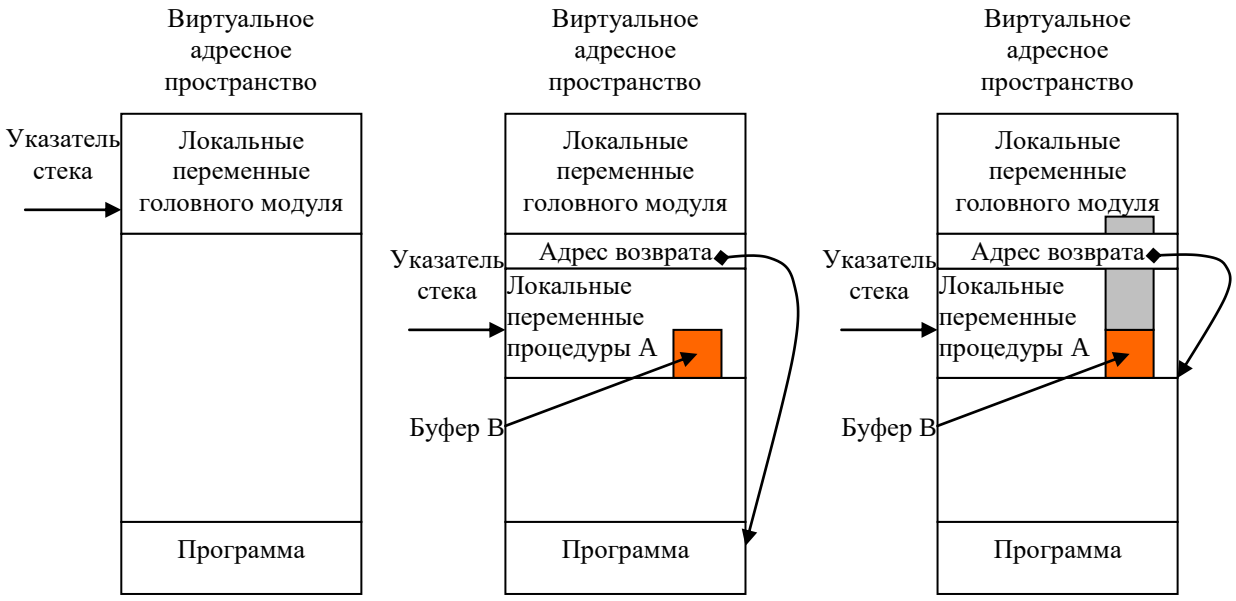


Рисунок 51 – Работает головная программа (а); вызывается процедура ***A*** (б); переполнение буфера (в)

Для распространения вируса, злоумышленник обычно внедряет его в какую-нибудь пиратскую копию программы и выкладывает её в Интернет. После чего пользователи начинают загружать программу на свой компьютер, где вирус размножается и выполняет действия, ради которых он и разрабатывался. На настоящий момент выделяют 7 основных видов вирусов.

1 Вирусы-компаньоны. Эти вирусы не заражают программу, а запускаются вместо какой-либо программы. Например, в системе MS-DOS или в командной строке Windows (или в пункте Пуск→Выполнить) пользователь вводит команду prog. При этом операционная система ищет сначала prog.com, а если его нет, то prog.exe. Поэтому автор вируса может назвать программу схожую с известной exe, но с расширением com и запустить его первым. Закончив выполняться, вирус запускает оригинал.

2 Вирусы, заражающие исполняемые файлы. Самый простой вирус записывает себя поверх исполняемой программы, поэтому их называют перезаписывающими. Однако их легко обнаружить при запуске программы. Паразитическими вирусами называются вирусы прицепляющиеся к программам, позволяя им нормально выполняться, что представлено на рисунках 52, б и в.

3 Резидентные вирусы. Если первые два типа вируса по выполнении собственных задач завершают существование в памяти, то резидентные всегда находятся в памяти. Типичный резидентный вирус перехватывает один из векторов прерывания (например, от устройства ввода-вывода или системного вызова), сохраняет старое значение в своей переменной и подменяет его адресом своей процедуры. При каждом срабатывании прерывания вирус инфицирует программы.

4 Вирусы, поражающие загрузочный сектор. Вирус сначала копирует исходное содержимое загрузочного сектора, чтобы иметь возможность загружать операционную систему. При загрузке компьютера вирус копирует себя в оперативную память, а после загружает операционную систему, оставаясь при этом резидентным в памяти. Вирус перехватывает себе все векторы прерываний (от принтера, диска, часов и т.д.) и работает через них, контролируя все системные вызовы.

5 Вирусы драйверов устройств. Вирус в этом случае инфицирует драйверы устройств, получая возможность перехватывать вектор системных прерываний.

6 Макровирусы. Макросы в Word и Excel также могут использоваться злоумышленниками для атаки операционных систем. Открытие документа с макросом вызывает выполнение его. А макрос может заражать другие документы Word, удалять файлы и т.д.

7 Вирусы, заражающие исходные тексты программ. Эти вирусы работают с исходными текстами программ, что делает их переносимыми для различных платформ.

Следует отличать вирусы от Интернет-червей: первые размножаются при запуске пользователем, а вторые используют дыры в операционной системе для самостоятельного проникновения и размножения.

Для борьбы с вирусами используются антивирусные программы, такие как DrWeb (http://www.drweb.com), Антивирус Касперского (http://www.kaspersky.ru) и многие другие. Также используются различные методы защиты от вирусов, описанные в ряде научных трудов, например [6].

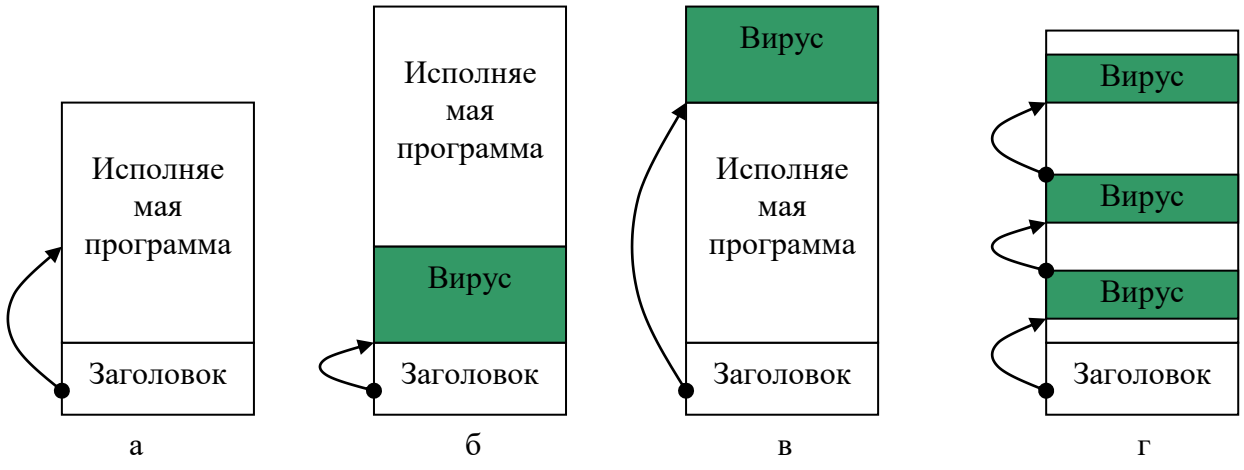


Рисунок 52 – Исполняемый файл (а); с вирусом в начале (б); с вирусом в конце (в); с вирусом, распределенным по свободным участкам программы (г)

Однако одними вирусами атаки снаружи операционной системы не ограничиваются [10, 11, 12]. Помимо всевозможных «дыр» в защите конкретных

операционных систем, сетевых устройств и межсетевых экранов существует ещё угроза атаки отказа в обслуживании или DOS-атака (Denial of Service), которая в основном используется для атаки Интернет-сервера. Атака такого рода препятствует или полностью блокирует ответы законным пользователям. Существует несколько типов DOS-атак:

- захват полосы пропускания. Атакующий занимает всю полосу пропускания сети, например, переправляя трафик с других сайтов;
- истощение ресурсов. Атака направлена на системные ресурсы – процессорное время, память и т.п., что приводит к сбою в работе операционной системы;
- ошибки программирования. Они заключаются в неспособности приложения, операционной системы или логической микросхемы обрабатывать исключительные ситуации. Это достигается передачей не корректных или не предусмотренных данных и команд;
- маршрутизация и атака DNS. Основываются на манипуляции записями таблицы маршрутизации, что приводит к отказу в обслуживании легитимных систем или сетей.

Механизмам атаки и защиты операционных систем посвящено множество трудов, в которых рассмотрены как конкретные системы, так и принципы в целом.

5.4 Защищенная операционная система

Очевидно, что операционную систему можно назвать защищенной, если она предусматривает средства защиты от основных классов угроз. Существует 2 подхода к созданию защищенных операционных систем: фрагментарный и комплексный.

При фрагментарном подходе вначале организуется защита от одной угрозы, затем от другой и т. д. При применении фрагментарного подхода подсистема защиты представляет собой набор разрозненных программных продуктов, как правило, от разных производителей. Эти программные средства работают

независимо друг от друга, при этом практически невозможно организовать их тесное взаимодействие [21].

При комплексном подходе защитные функции вносятся в операционную систему непосредственно на этапе проектирования архитектуры.

Несмотря на наличие программных средств защиты, для комплексной зашиты операционной системы, нужно использовать и организационные меры. Ниже перечислены основные административные меры защиты [21]:

- постоянный контроль корректности функционирования операционной системы, особенно ее подсистемы защиты;
- организация и поддержание адекватной политики безопасности;
- инструктирование пользователей операционной системы о необходимости соблюдения мер безопасности при работе с операционной системой и контроль за соблюдением этих мер;
- регулярное создание и обновление резервных копий программ и данных операционной системы;
- постоянный контроль изменений в конфигурационных данных и политике безопасности операционной системы.

Выбор и поддержание адекватной политики безопасности являются одной из наиболее важных задач администратора. Чем лучше защищена система, тем труднее с ней работать пользователям и администраторам. Поэтому необходимо стремиться к оптимальной политике безопасности. Этапы формирования адекватной политики безопасности: анализ угроз, формирование требований, формализация задачи, реализация политики безопасности, поддержка.

Специальных стандартов обеспечения защищенности операционных систем не существует. Для оценки защищенности операционной системы используются стандарты, разработанные для компьютерных систем в целом [21].

5.5 Подсистема защиты операционной системы

Если взять ряд операционных систем и проанализировать возможности

подсистем защиты, то можно определить следующие основные задачи, которая должна решать подсистема защиты ОС:

- обеспечение различных способов аутентификации и идентификации пользователей;
- разграничение доступа пользователей к данным и объектам операционной системы;
- журналирование или аудит операционной системы;
- создание, настройка и управление политикой безопасности операционной системы и групп операционных систем;
- обеспечение встроенных средств шифрования (например, EFS при NTFS);
- поддержка брандмауэра (или возможности установить сторонний брандмауэр).

Способы аутентификации были описаны в параграфе 5.2.

Разграничение доступа пользователей завязано на следующих понятиях, описанных ниже на основе данного источника [21].

Объектом доступа (или просто объектом) называют любой элемент ОС, доступ к которому пользователей и других субъектов доступа может быть произвольно ограничен.

Методом доступа к объекту называется операция, определенная для объекта. Тип операции зависит от объектов.

Субъектом доступа называют любую сущность, способную инициировать выполнение операций над объектами (обращаться к объектам по некоторым методам доступа).

Правом доступа к объекту называют право на выполнение доступа к объекту по некоторому методу или группе методов.

Разграничением доступа субъектов к объектам является совокупность правил, определяющих для каждой тройки субъект – объект - метод, разрешен ли доступ данного субъекта к данному объекту по данному методу.

Существуют две основные модели разграничения доступа: избирательное

(дискреционное) и полномочное (мандатное) разграничение доступа.

В большинстве операционных систем реализована дискреционная модель разграничения доступа, которую можно представить в виде матрицы, строки которой соответствуют субъектам, а столбы – объектам. В соответствующих ячейках данной матрицы определяются права доступа к субъекта к объекту. Достоинством данной модели является возможность гибкой настройки прав доступа к объектам. К недостаткам можно отнести значительный объем работ по назначению прав доступа при большом количестве субъектов и объектов. Для уменьшения трудоемкости работы по назначению прав доступа в операционных системах может использоваться механизм задания прав доступа по умолчанию, например, все запрещено, что явно не разрешено.

Также часто применяется ролевая модель доступа, которая является модификацией дискреционной. В данном случае, пользователи, выполняющие в организации одну роль, объединяются в группы. Права доступа назначаются тогда не каждому пользователю, а только группе.

В дискреционной модели права доступа к объекту может назначать владелец ресурса. Это может привести к случаям, когда владелец объекта случайно или преднамеренно установит не правильные права доступа для других субъектов.

Данного недостатка нет в мандатной модели, когда доступ определяется на основе меток конфиденциальности объектов и субъектов. При совпадении уровня допуска пользователя с меткой конфиденциальности ресурса, то субъект может совершать любые действия. Если у субъекта уровень допуска выше чем у объекта, то он может выполнять только операцию чтения, если меньше – то все операции запрещены. Причем необходимо отметить, что все вновь созданные объекты (например, файлы) будут иметь метку доступа владельца. Владелец при этом не может изменить данную метку. К недостатку данной модели относится сложность задачи точной классификации субъектов и объектов в небольшое количество уровней допуска. Так как мандатная модель разграничения доступа обеспечивает более высокий уровень защищенности, то ФСТЭК требует применение его в классах защищенности СВТ 4 и выше.

Процедура аудита заключается ведение журналов событий, фиксирующих важные события операционной системы. Данные аудита используются для расследования инцидентов информационной безопасности. Основными требованиями к политике аудита являются следующие:

- добавлять записи в журнал аудита может только ОС. Если предоставить это право какому-то физическому пользователю, то он получит возможность компрометировать других пользователей, добавляя в журнал аудита ложные записи;

- редактировать или удалять отдельные записи в журнале аудита не может ни один субъект доступа, в том числе и сама ОС;

- просматривать журнал аудита могут только пользователи, обладающие соответствующей привилегией;

- очищать журнал аудита могут только пользователи-аудиторы. После очистки журнала в него автоматически вносится запись о том, что журнал аудита был очищен, с указанием времени очистки журнала и имени пользователя, очистившего журнал. ОС должна поддерживать возможность сохранения журнала аудита перед очисткой в другом файле;

- при переполнении журнала аудита ОС аварийно завершает работу («зависает»). После перезагрузки работать с системой могут только аудиторы. ОС переходит к обычному режиму работы только после очистки журнала аудита[21].

5.6 Структура системы безопасности Windows 7

В структуру операционной системы (ОС) входят следующие ключевые элементы с точки зрения безопасности:

- возможность защищённого входа в систему с аутентификацией пользователей;
- контроль доступа по категориям пользователей;
- аудит, т.е. независимая проверка с целью выражения мнения о достоверности.

Это далеко не полный список элементов, однако они наиболее важные с точки

зрения безопасности – аутентификация, авторизация и аудит. Они были встроены в ОС Windows NT с самого начала. Во всех системах этого семейства, включая и современные, эти возможности реализованы с помощью подсистемы безопасности (рисунок 65).

Главным элементом системы безопасности является менеджер безопасности режима ядра (SRM), исполняющий сложный механизм безопасности Windows, удовлетворяющий требованиям класса C2 Оранжевой книги Министерства обороны США. Помимо этого в режиме пользователя работают следующие важные компоненты.

Winlogon – компонент операционной системы Windows, отвечающий за вход в систему и т.д. Winlogon обрабатывает нажатие Ctrl-Alt-Del и Ctrl-Shift-Esc. В ходе запуска ОС Winlogon запускает LSASS и Services.exe.

Lsass – подсистема полномочий локальной безопасности, т.е. часть ОС отвечающая за авторизацию локальных пользователей отдельного компьютера. При получении полного доступа к данному сервису злоумышленник может получить полные права для доступа к компьютеру. Поэтому способы шифрования и передачи данных для авторизации между компонентами не документируется.

Сервер LSA отвечает за вход в систему. Служба Net Logon автоматически запускается, только когда к домену подключается компьютер или контроллер домена.

SAM – RPC-сервер Windows, оперирующий базой данных учетных записей.

Active Directory – LDAP-совместимая реализация интеллектуальной службы каталогов корпорации Microsoft для операционных систем семейства Windows NT. Active Directory позволяет администраторам использовать групповые политики (GPO) для обеспечения единообразия.

Суть рисунка 53 сводится к тому, что в Windows реализован диспетчер системы безопасности, который выполняется в высоко-привилегированном режиме ядра и проверяет все запросы ресурсов из кода, исполняющегося в пользовательском режиме.

Диспетчер SRM выступает в роли хранителя ресурсов Windows. Почти весь

контроль доступа к ресурсам Windows осуществляется по отношению к элементам системы безопасности. Элементами системы безопасности Windows являются: пользователи, группы, компьютеры.

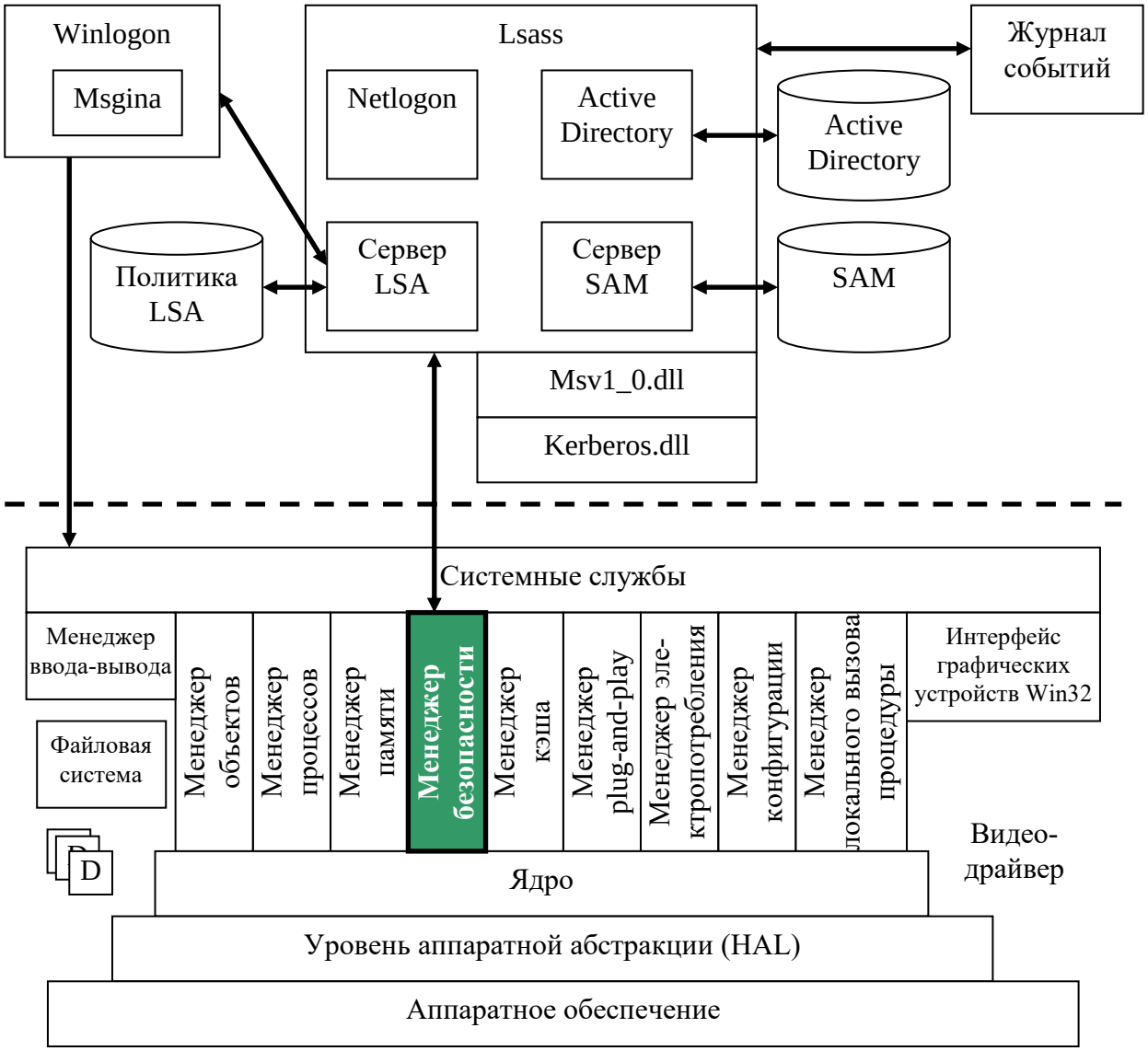


Рисунок 53 – Подсистема безопасности Windows

Важными понятиями являются аутентификация и авторизация. ОС должна определить, что она работает с действительным элементом системы безопасности. Это делается посредством аутентификации. Простейший пример – вход пользователя в систему Windows с консоли. Пользователь нажимает клавиши <CTRL+ALT+DEL>, после чего вводит свои логин и пароль. Программа защиты входа в систему обрабатывает введенные данные с помощью компонентов,

работающих в пользовательском режиме, как показано на рисунке 53 (программы Winlogon и LSASS). Если аутентификация проходит успешно, программа Winlogon создаст маркер доступа (набор атрибутов пользователя или процесса), который присваивается сеансу работы пользователя и используется при любой последующей попытке доступа к ресурсам.

Маркер доступа содержит список всех идентификаторов SID, связанных с учетной записью пользователя, включая идентификатор SID самой учетной записи, идентификаторы SID всех групп, в которые добавлена эта учетная запись, и идентификаторы специальных групп, к которым относится данный пользователь (например, Domain Admins или INTERACTIVE). Чтобы узнать, какие идентификаторы SID связаны с сеансом работы, можно воспользоваться программой whoami, которая входит в пакет дополнительных программ Windows. Результат выполнения данной команды приведен ниже:

```
C:\>whoami /user /groups

USER INFORMATION
----------------

User Name         SID
================= ============================================
test\administrator S-1-5-21-351884573-2638605479-2349113266-500
GROUP INFORMATION
-----------------
Group Name                           Type             SID          Attributes
==================================== ================ ============
Everyone                             Well-known group S-1-1-0      Mandatory group,
Enabled by default, Enabled group
BUILTIN\Administrators               Alias            S-1-5-32-544 Mandatory group,
Enabled by default, Enabled group, Group owner
BUILTIN\Users                        Alias            S-1-5-32-545 Mandatory group,
Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE             Well-known group S-1-5-4      Mandatory group,
Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users     Well-known group S-1-5-11     Mandatory group,
Enabled by default, Enabled group
NT AUTHORITY\This Organization       Well-known group S-1-5-15     Mandatory group,
Enabled by default, Enabled group
```

```
LOCAL                              Well-known group S-1-2-0      Mandatory group,
Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication Well-known group S-1-5-64-10  Mandatory group,
Enabled by default, Enabled group
```

Синтаксис использования whoami имеет следующий вид:

WHOAMI [/UPN | /FQDN | /LOGONID]

WHOAMI { [/USER] [/GROUPS] [/PRIV] } [/FO <Σ«α¼áΓ>] [/NH]

WHOAMI /ALL [/FO <формат>] [/NH]

Параметры:

/UPN Отображение имени пользователя в формате имени участника-пользователя (UPN).

/FQDN Отображение имени пользователя в формате полного доменного имени (FQDN).

/USER Отображение сведений о текущем пользователе вместе с идентификатором безопасности (SID).

/GROUPS Отображение для текущего пользователя членства в группах, типа учетной записи, идентификаторов безопасности (SID) и атрибутов.

/PRIV Отображение привилегий безопасности текущего пользователя.

/LOGONID Отображение идентификатора текущего пользователя.

/ALL Отображение имени пользователя, членства в группах, идентификаторов безопасности (SID) и привилегий для токена доступа текущего пользователя.

/FO <формат> Формат вывода. Допустимые значения TABLE, LIST, CSV. Заголовки столбцов в формате CSV не отображаются. Формат по умолчанию: TABLE.

/NH Указывает, что строка заголовков столбцов не отображается при выводе.

Допускается только для форматов TABLE и CSV.

/? Вывод справки по использованию.

Когда пользователь пытается получить доступ к некоторому ресурсу, например к файлу, программа SRM сравнивает его маркер доступа со списком разграничительного контроля доступа DACL объекта. Список DACL содержит идентификаторы SID, для которых разрешен доступ к объекту, и разрешенный тип доступа (чтение, запись, выполнение и т.п.). Если один из идентификаторов SID учетной записи пользователя совпадает с идентификатором SID из списка DACL, то пользователь получает доступ в соответствии с условиями, указанными в DACL. Эти действия отражены на рисунке 54.

В современно мире аутентификации непосредственно на одном персональном компьютере недостаточно, поэтому используется аутентификация по сети, что является потенциально более опасным механизмом.

В системах семейства Windows NT в основном используют аутентификацию с запросом и подтверждением, при которой сервер передает клиенту случайное число (запрос), клиент затем обрабатывает полученное число с помощью функции хеширования, используя хешированный пароль пользователя, и возвращает новое хешированное значение (ответ) серверу. После этого сервер берет свою копию хешированного пароля пользователя из локальной базы данных SAM или Active Directory, хеширует отправленный им запрос и сравнивает полученное значение с ответом клиента. Таким образом, при аутентификации в системах семейства Wndows NT пароли не передаются по сети даже в зашифрованном виде (рисунок 55).

На всех компьютерах под управлением Windows NT информация об именах учетных записей и паролях содержится в базе данных диспетчера системы защиты (служба SAM – Security Access Manager). Пароли хранятся в зашифрованном виде, их невозможно расшифровать, пользуясь известными методами (хотя зашифрованное значение можно подобрать). Процедура шифрования называется односторонней функцией (ОСФ) или алгоритмом хеширования, значение хеш-функции расшифровать нельзя. Программный интерфейс для доступа клиентов к

серверу реализован в виде функций, содержащихся в DLL-библиотеке samlib.dll.

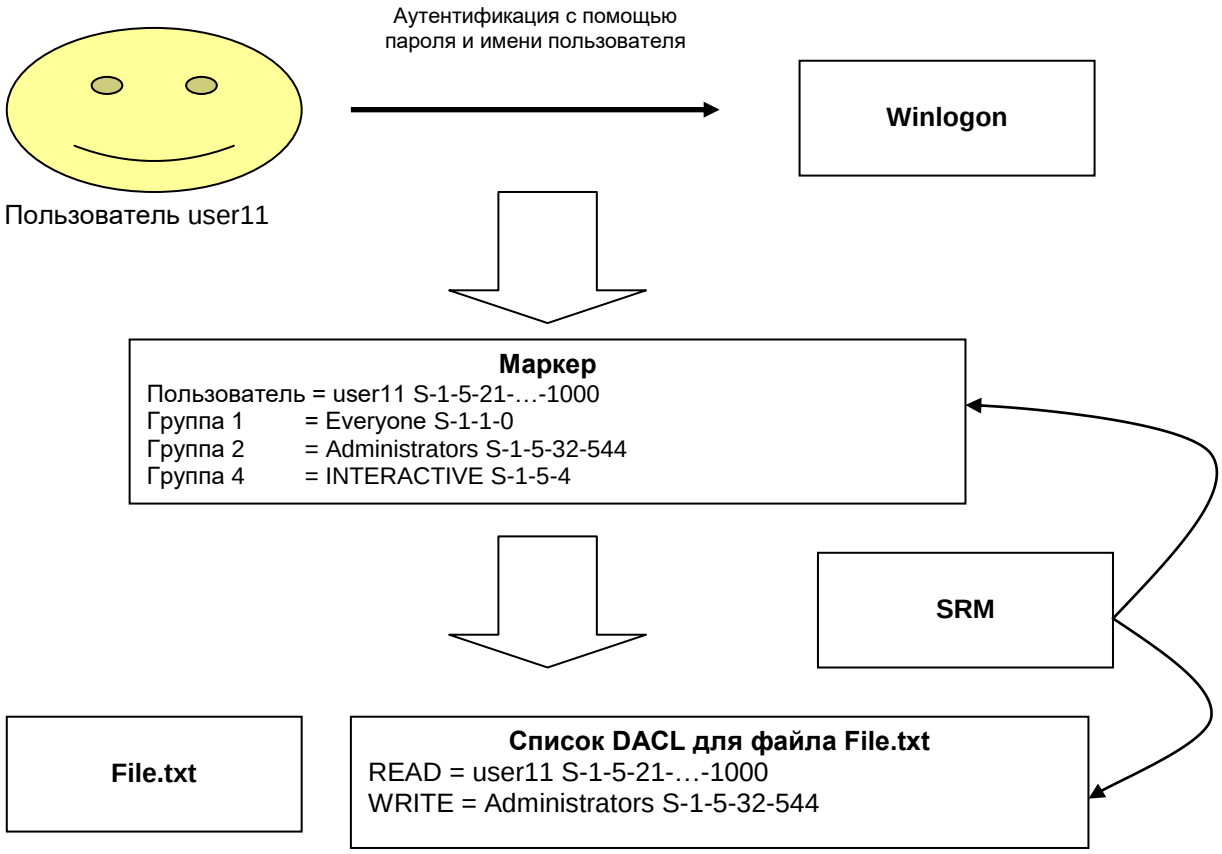


Рисунок 54 – Процедура аутентификации при помощи пароля и имени

SAM выполняет следующие задачи:

– идентификация субъектов;

– проверка пароля, авторизация (участвует в процессе входа пользователей в систему);

– хранит статистику (время последнего входа, количества входов, количества некорректных вводов пароля);

– хранит настройки политики учетных записей и приводит их в действие (политика паролей и политика блокировки учетной записи);

– хранит логическую структуру группировки учетных записей (по группам, доменам);

– контролирует доступ к базе учетных записей;

– предоставляет программный интерфейс для управления базой учетных записей.

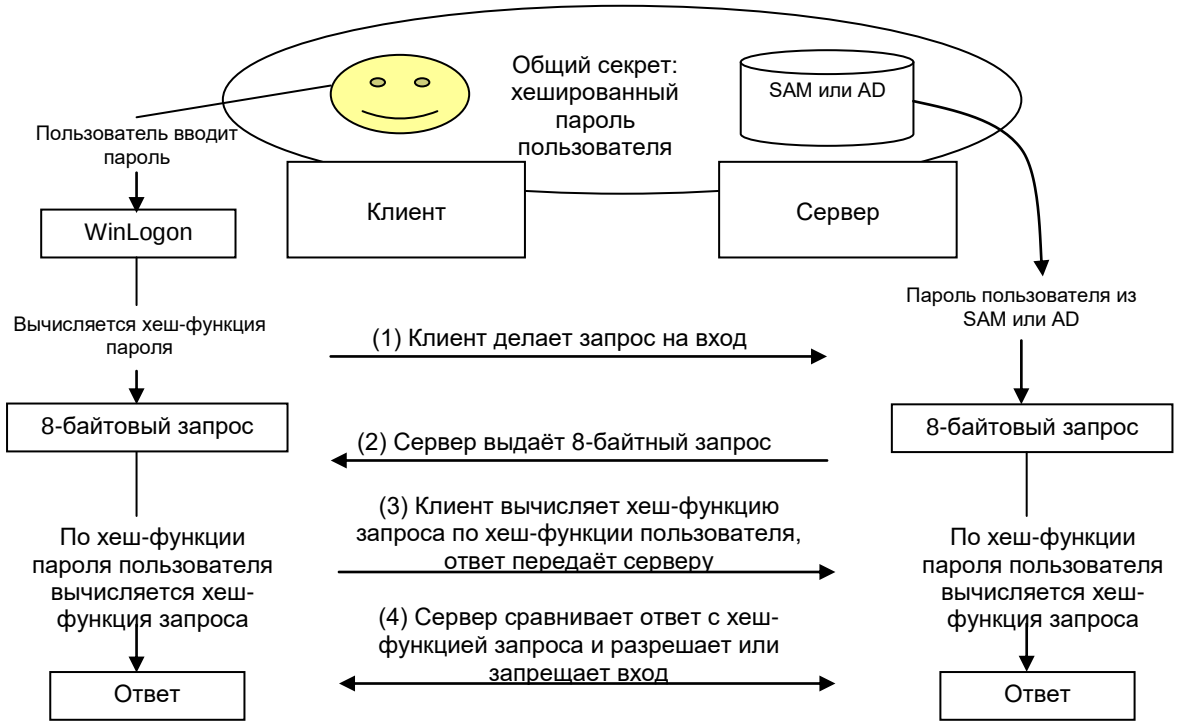


Рисунок 55 – Аутентификации по сети

База данных SAM хранится в реестре (в ключе HKEY_LOCAL_MACHINE\SAM\SAM), доступ к которому запрещен по умолчанию даже администраторам.

На контроллерах доменов имя учетной записи и хешированный пароль хранятся службой Active Directory.

Начиная с версии NT4, в пакете обновления 3 фирма Microsoft обеспечила возможность использования еще одного уровня шифрования для хешированных паролей SAM, который называется SYSKEY. SYSKEY вычисляет случайный 128-разрядныи ключ и этим ключом еще раз шифрует хешированные пароли (только хеш-функции, а не сам файл SAM).

Рисунок 56 – Вызов команды syskey

Ключ шифрования SYSKEY может храниться в одном из трех режимов.

Режим 1. Хранится в реестре и предоставляется автоматически во время загрузки (настройка по умолчанию).

Режим 2. Хранится в реестре, но заблокировано паролем, который необходимо ввести во время загрузки.

Режим 3. Хранится на гибком диске и должно быть предоставлено во время загрузки. Выбор этих режимов показан на следующем рисунке.

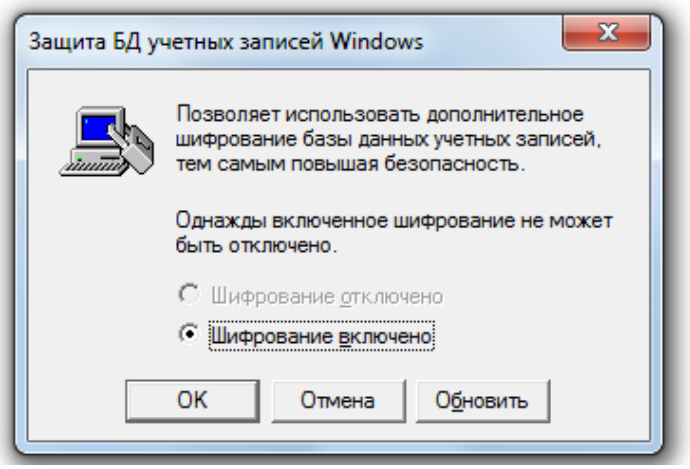


Рисунок 57 – Результат вызова команды syskey

Если в окне, показанном на рисунке 57 выбрать кнопку «Обновить», то на экране появится следующее окно (рисунок 58)

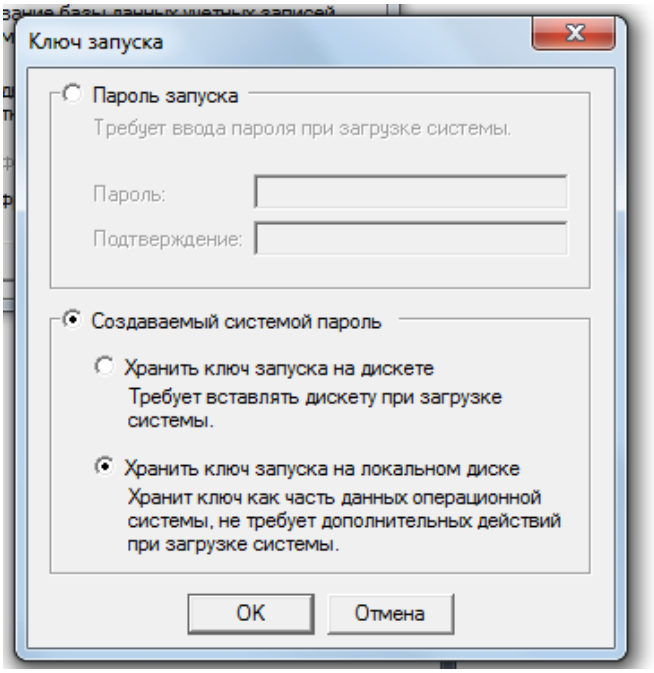


Рисунок 58 – Окно настроек syskey

Первый два варианта хранения пароля считаются более надежными, так секретную информацию нельзя извлечь непосредственно из компьютера. Таким образом, программа SYSKEY позволяет защитить компьютер от атаки перебора паролей пользователей.

5.7 Контрольные вопросы

1. Перечислите основные задачи и угрозы безопасности.
2. Что такое аутентификация? Чем аутентификация отличается от идентификации?
3. Каковы недостатки аутентификации по многоразовому паролю?
4. В чём заключается вариант использования защиты по паролю «отклик-отзыв»?
5. Поясните, как работает атака с использованием переполнения буфера. Почему данная атака возможна?
6. Какой ущерб могут причинить вирусы? Перечислите существующие виды вирусов?
7. Чем вирусы отличаются от Интернет-червей?
8. Что такое DOS-атака? Какие существуют типы DOS-атак?
9. Перечислите методы административной защиты операционной системы.
10. Какие две модели разграничения доступа существуют?

Заключение

В учебном пособии рассмотрены основные понятия и принципы работы подсистем операционных систем. В первой главе представлен материал о структуре операционных систем, основных видах их архитектур. Описано реализация основных подсистем современных ОС Windows и Linux. Вторая глава посвящена объектам исполнения: процессам и потокам. Определены принципы их реализации. Большое внимание уделено вопросам межпроцессного взаимодействия и алгоритм планирование времени процессора.

Вопросы управления памятью рассмотрены в третьей главе данной работы. Рассматриваются различные способы организации памяти, такие как страничная, сегментная, сегментно-страничная, определены их достоинства и недостатки. Описаны принципы реализации виртуальной памяти.

Организация системы ввода-вывода современных операционных систем представлена в четвертом разделе работы. Определена иерархическая структура данной системы, выделены основные компоненты, начиная с контроллеров устройств до программ пользовательского уровня. Выделены особенности различных способов обмена данными с внешними устройствами, включая ввод-вывод с использованием режимов прямого доступа и на основе системы прерываний. Также рассмотрены вопросы организации файловых систем на примере FAT и NTFS. Основные вопросы обеспечения информационной безопасности операционных систем представлены в пятой главе.

Таким образом, представленный в данной работе материал позволяет сформировать базовые знания в области устройства, функционирования и обеспечения ИБ современных операционных систем. Полученные знания помогут более эффективно разрабатывать программные средства, эксплуатировать и администрировать вычислительные системы. Материалы учебного пособия рекомендуется использовать при проведении лекционных и практических занятий по дисциплине «Операционные системы».

Список использованных источников

1 Аблязов, Р. З. Программирование на ассемблере на платформе x86-64 / Р.З. Аблязов. – М.: ДМК Пресс, 2011. – 304 с

2 Аралбаев, Т.З. Контроль и управление доступом в АСУ ТП на основе биометрических характеристик пользователя / Т.З. Аралбаев, А.Г. Африн. – Уфа: Гилем, 2008. – 124 с.

3 Бурдонов, И.Б. Операционные системы реального времени / И.Б. Бурдонов, А.С. Косачев, В.Н. Пономаренко // Препринт Института системного программирования РАН. – 2006. – № 14.

4. Дейтел, Х.М. Операционные системы. Основы и принципы. / Х.М. Дейтел, П.Дж. Дейтел. – М. : Бином, 2006. – 1024 с.

5 Зубков, С.В. Assembler. Для DOS, Windows и Unix / С.В. Зубков. – М.: ДМК, 2006. – 608 с.

6 Казарин, О.В. Безопасность программного обеспечения компьютерных систем / О.В. Казарин. – М.: МГУЛ, 2003. – 212 с.

7 Коньков, К.А. Основы организации операционных систем Microsoft Windows [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/os/osmswin/ .

8 Кэрриэ, Б. Криминалистический анализ файловых систем / Б. Кэрриэ. – СПб.: Питер, 2007. – 480с.

9 Лав Р. Ядро Linux: описание процесса разработки, 3-е изд. : Пер. с англ. / Р. Лав. — М. : ООО «И.Д. Вильямс», 2013. — 496 с.

10 Мак-Клар, С. Секреты хакеров. Безопасность сетей – готовые решения / С. Мак-Клар, Дж. Скембрей, Дж. Курц. – М. : Издательский дом «Вильямс», 2002. – 736с.

11 Мак-Клар, С. Секреты хакеров. Безопасность Windows 2003 — готовые решения / С. Мак-Клар, Дж. Скембрей, Дж. Курц. – М.: Издательский дом «Вильямс», 2004. – 512 с

12 Мак-Клар, С. Секреты хакеров. Проблемы и решения сетевой защиты / С. Мак-Клар, Дж. Скембрей, Дж. Курц. – М.: Лори, 2001. – 464 с.

13 Михайлов, Д. Файловая система NTFS. [Электронный ресурс] / Д. Михайлов. –

Режим доступа: http://www.ixbt.com/storage/ntfs.html .

14 Михайлов, Д. Надежность дисковой системы NT. [Электронный ресурс] / Д. Михайлов. – Режим доступа: http://www.ixbt.com/storage/ntfs2.html .

15 Немюгин, С.А. Параллельное программирование для многопроцессорных вычислительных систем / С.А. Немюгин, О.Л. Стесик. – СПб.: БХВ-Петербург, 2002. – 400с.

16 Олифер, В.Г. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер. – 2002. – 544 с

17. Руководство по безопасности Windows® 7 [Электронный ресурс]. – Режим доступа: http://technet.microsoft.com/ru-ru/library/ee914622.aspx .

18 Саймон, Р. Windows 2000 API. Энциклопедия программиста : пер. с англ. / Р. Саймон. – СПб. : ООО «ДиаСофтЮП», 2002. – 1088 с.

19 Таненбаум, Э. Архитектура компьютера : пер. с англ. / Э. Таненбаум. – 6 изд. – СПб: Питер. – 2016. – 816 с.

20 Таненбаум, Э. Современные операционные системы. / Э. Таненбаум. – 4-е изд. – СПб.: Питер, 2015. — 1120 с.

21 Шаньгин В.Ф. Информационная безопасность / В.Ф. Шаньгин. – М.: ДМК Пресс, 2014. – 702 с.

22 Coffman, E.G., Elphick, M.J., and Shoshani, A.: «System Deadlocks», Computing Surveys, vol. 3, pp. 67-78, June 1971.

23 Dijkstra, E.W.: «Cooperating Sequential Processes», in Programming Languages, Genuys, F. (Ed.), London: Academic Press, 1965.

24 Peterson, G.L.: «Myths about the Mutual Exclusion Problem», Information Processing Letters, vol. 12, pp. 115-116, June 1981.