ГОУ «ПРИДНЕСТРОВСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Т. Г. ШЕВЧЕНКО»

Физико-технический институт
Факультет среднего профессионального образования
(технический колледж им. Ю. А. Гагарина)
Кафедра интегрированных компьютерных технологий и систем

ОПЕРАЦИОННЫЕ СИСТЕМЫ И СРЕДЫ

Учебное пособие

Тирасполь

Издательство Приднестровского университета

2025

УДК 004.451(075.32)
ББК 3972.111я723
О60

Составители:
О. М. Фурдуй, доцент
Т. С. Новакова, ст. преподаватель
Е. Г. Яковенко, ст. преподаватель

Рецензенты:
Л. И. Гончарук, заместитель руководителя по учебной работе Тираспольского техникума информатики и права
Ю. С. Столяренко, кандидат технических наук кафедры информационных технологий ФТИ (Приднестровский государственный университет им. Т. Г. Шевченко)

О60 **Операционные системы и среды** : учебное пособие [Электронный ресурс] / ГОУ «Приднестровский государственный университет им. Т. Г. Шевченко» ; Физико-технический институт ; Факультет среднего профессионального образования (технический колледж им. Ю. А. Гагарина) ; составители : О. М. Фурдуй, Т. С. Новакова, Е. Г. Яковенко. – Тирасполь : Изд-во Приднестр. ун-та, 2025. – 78 с.

Системные требования: CPU (Intel/AMD) 1,5ГГц/ОЗУ 2ГГб/HDD 450Мб/1024*768/Windows 7 и старше/Internet Explorer 11/Adobe Acrobat Reader 6 и старше.

Подготовлено в соответствии с учебным планом специальности: 09.02.01 «Компьютерные системы и комплексы». Учебное пособие отличается от известных учебников рациональной компоновкой учебного материала, содержит теоретический материал для более глубокого усвоения новых понятий, в дальнейшем для выполнения лабораторных работ по одноименному предмету. Адресовано обучающимся ФСПО (Технического колледжа им. Ю. А. Гагарина) по специальности 09.02.01 «Компьютерные системы и комплексы».

УДК 004.451(075.32)
ББК 3972.111я723

Рекомендовано Научно-методическим советом ПГУ им. Т. Г. Шевченко

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире информационных технологий операционные системы участвуют в нашей жизни гораздо больше, чем мы подозреваем. Они управляют аппаратными ресурсами компьютеров, смартфонов, серверов и других устройств, обеспечивая эффективную работу приложений и пользовательский комфорт. В данном учебном пособии описаны основные концепции и принципы работы операционных систем, их архитектура, функции и особенности различных сред выполнения программ. Цель пособия – дать прочную теоретическую базу и практические навыки для понимания и работы с современными операционными системами, что важно для профессионального роста в области информационных технологий.

Операционные системы – это основа цифрового мира, и их глубокое понимание открывает широкие горизонты возможностей для развития в IT-сфере.

Операционные системы – это не просто набор программ, управляющих аппаратными компонентами и приложениями. Это сложные, эволюционирующие экосистемы, которые лежат в основе всей цифровой инфраструктуры современного мира. Их роль выходит за рамки простого выполнения команд – они обеспечивают баланс, безопасность, устойчивость и эффективность работы сложных систем.

В современном мире операционные системы сталкиваются с множеством вызовов: обработка огромных объёмов данных, поддержка распределённых и облачных вычислений, обеспечение кибербезопасности и поддержки новых типов устройств – от мобильных телефонов и IoT-устройств до суперкомпьютеров. Их дизайн становится всё более гибким и адаптивным, а архитектуры – более модульными и масштабируемыми.

Понимание внутренней структуры ОС, её компонентов – ядра, драйверов, системных вызовов, менеджеров ресурсов – позволяет разрабатывать более эффективные алгоритмы, создавать новые приложения и повышать безопасность информационных систем. Не менее важно то, что знание принципов работы ОС помогает разрабатывать инновационные решения для оптимизации использования вычислительных ресурсов, снижения энергопотребления и повышения устойчивости систем к сбоям и атакам. Всё это делает операционные системы ключевым элементом глобальной цифровой трансформации.

В конечном итоге, изучение операционных систем – это открытие двери в сложный, захватывающий и постоянно развивающийся мир технологий.

РАЗДЕЛ 1. ВВЕДЕНИЕ. ОПЕРАЦИОННЫЕ СИСТЕМЫ (ОС)

Тема 1.1. Определение, задачи и функции ОС. Структура вычислительной системы

ОС в различных аспектах. Концепция операционной среды

Операционная система (ОС) представляет собой совокупность управляющих и обрабатывающих программ, которые выполняют две ключевые *функции:* они служат промежуточным звеном между компьютерным оборудованием и пользователем, выполняющим свои задачи, и обеспечивают эффективное использование ресурсов вычислительной системы, а также организуют надежный процесс вычислений.

ОС изолирует аппаратное обеспечение компьютера от пользовательских приложений, предоставляя интерфейсы для их взаимодействия. Например, пользователи и их программы обращаются к компьютеру через интерфейсы операционной системы.

К числу известных операционных систем относятся: UNIX, OS/2, Windows, Linux, QNX, MacOS, BeOS.

Структуру вычислительной системы можно описать как комбинацию аппаратного и программного обеспечения.

Аппаратное обеспечение включает в себя:

– процессор;

– оперативную память;

– монитор;

– накопители и другие устройства.

Программное обеспечение делится на две категории:

Прикладное программное обеспечение: это разнообразные пользовательские приложения, включая игры, текстовые редакторы и прочее.

Системное программное обеспечение: содержит набор программ, который поддерживает функционирование и разработку прикладных программ (табл. 1.).

В системе системного ПО выделяются:

Базовое ПО: минимальный набор программ, необходимых для работы компьютера, включая операционную систему и ее оболочки.

Сервисное ПО: программы и комплексы, которые расширяют функции базового ПО и улучшают удобство работы пользователю. Сервисное ПО также включает средства разработки, такие как редакторы, компиляторы и интерпретаторы.

Таблица 1

Положение операционной системы в общей структуре компьютера

<table>
<tr><td>Веб-браузер</td><td>Базы данных</td><td>Игры</td><td>Прикладное ПО</td></tr>
<tr><td>Компиляторы</td><td>Редакторы</td><td>Интерпретаторы</td><td rowspan="2">Системное ПО</td></tr>
<tr><td colspan="3">Операционная система</td></tr>
<tr><td colspan="3">Машинный язык</td><td rowspan="3">Аппаратура</td></tr>
<tr><td colspan="3">Микроархитектура</td></tr>
<tr><td colspan="3">Физические устройства</td></tr>
</table>

Таким образом, операционная система является основополагающим компонентом системного программного обеспечения.

Классификация ОС

Существует несколько подходов к классификации операционных систем. Вот некоторые из *критериев:*

1. Поддержка многопользовательского режима.

На основании количества одновременно работающих пользователей ОС делятся на:

– Однопользовательские (например, MS-DOS, Windows 3.x).

– Многопользовательские (например, Windows NT, Unix), которые оснащены механизмами защиты данных для каждого пользователя.

2. Многопроцессорная обработка.

В соответствии с этой характеристикой ОС можно разделить на:

– Однопроцессорные.

– Многопроцессорные, которые поддерживают мультипроцессирование и используют более сложные алгоритмы управления ресурсами (такие как Linux, Solaris, Windows NT). Многопроцессорные системы могут включать два и более центральных процессоров, что позволяет им выполнять команды параллельно.

Многопроцессорные ОС далее классифицируются на:

– Симметричные, где несколько одинаковых процессоров равнозначно подключены к общей памяти и периферийным устройствам, выполняя одни и те же задачи.

– Асимметричные, в которых процессоры имеют разные уровни приоритета (например, один главный процессор и несколько подчиненных), при этом их загрузка и задания определяются главным процессором.

3. Работа в режиме реального времени.

Специальные ОС предназначены для работы в режиме реального времени.

Системы реального времени должны реагировать на внешние события в строго установленные сроки и обработка данных должна происходить быстрее, чем они поступают. Эти системы применяются для управления

техническими объектами или технологическими процессами, и их архитектура может не включать виртуальную память, что исключает непредсказуемые задержки.

4. Реализация многозадачности.

По количеству одновременно выполняемых задач операционные системы делятся на:

– Многозадачные (например, Unix, OS/2, Windows), которые полностью реализуют мультипрограммный режим.

– Однозадачные (например, MS-DOS).

Многозадачность

Особенности реализации многозадачности в системах пакетной обработки

Мультипрограммирование, также известное как многозадачность, представляет собой метод организации вычислительного процесса, при котором несколько программ попеременно выполняются на одном процессоре.

Критерии эффективности вычислительных систем

– Пропускная способность: количество задач, обрабатываемых системой в единицу времени.

– Удобство работы пользователей: насколько комфортно взаимодействовать с системой.

– Реактивность системы: способность системы выдерживать заранее заданные (возможно, очень короткие) интервалы между запуском программы и получением результата.

В зависимости от выбранных критериев эффективности операционные системы классифицируются на системы пакетной обработки, системы разделения времени и системы реального времени.

Особенности систем пакетной обработки

Основной целью систем пакетной обработки является максимизация пропускной способности, то есть решение наибольшего числа задач за единицу времени. Для достижения этой цели применяется следующая схема:

1. В начале работы формируется пакет заданий, каждое из которых содержит требования к ресурсам системы.

2. Из этого пакета формируется мультипрограммная смесь, которая позволяет одновременно выполнять несколько задач. Задачи, как правило, должны предъявлять различные требования к ресурсам, чтобы обеспечить сбалансированную загрузку всех устройств машины. Например, рационально комбинировать вычислительные задачи и задачи с интенсивным вводом-выводом.

Выбор нового задания для выполнения зависит от текущей загрузки системы, то есть выбирается наиболее «выгодное» задание.

В системах пакетной обработки переключение процессора между задачами инициируется самой активной задачей, например, когда она отказывается от процессора для выполнения операции ввода-вывода. Это может привести к ситуации, когда одна задача задерживает процессор на длительное время, что делает выполнение интерактивных задач невозможным.

Взаимодействие пользователя с системой ограничивается подачей задания диспетчеру-оператору, а результаты он получает только по завершении обработки всего пакета заданий.

Достоинства

– Увеличение эффективности работы аппаратуры.

Недостатки

– Снижение эффективности взаимодействия с пользователем.

Многозадачность в системах разделения времени

Мультипрограммирование, или многозадачность, также встречается в системах разделения времени, где несколько программ могут одновременно выполняться, но с улучшенной реакцией на действия пользователя.

Критерии эффективности вычислительных систем остаются прежними:

– Пропускная способность

– Удобство работы пользователей

– Реактивность системы

В зависимости от выбранных критериев, операционные системы могут делиться на три типа: пакетной обработки, разделения времени и реального времени.

Особенности реализации многозадачности в системах разделения времени

Целью систем разделения времени является повышение удобства и эффективности работы пользователя. В таких системах пользователям (или одному пользователю) предоставляется возможность интерактивно взаимодействовать сразу с несколькими приложениями. Для этого каждое приложение должно регулярно получать возможность «общения» с пользователем.

В системах разделения времени операционная система принудительно приостанавливает приложения, не дожидаясь, когда они сами освободят процессор. Каждому приложению попеременно выделяется квант процессорного времени, что позволяет пользователям взаимодействовать с запущенными программами в реальном времени.

Поскольку каждой задаче выделяется лишь квант времени, ни одна из них не может долго занимать процессор, что обеспечивает приемлемое время отклика. Если квант выбран достаточно маленьким, у всех пользовате-

лей, работающих на одной машине, создается впечатление, что каждый из них использует машину единолично.

Недостатки систем разделения времени:

– Системы разделения времени имеют меньшую пропускную способность по сравнению с системами пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не наиболее «выгодная» для системы.

– Производительность снижается из-за увеличенных накладных расходов на частое переключение процессора между задачами.

Достоинства систем разделения времени:

– Повышение удобства и эффективности работы пользователя.

– Мультипрограммное выполнение интерактивных приложений увеличивает пропускную способность системы, поскольку, пока одно приложение ожидает ввода от пользователя, другие могут продолжать свою работу.

Требования к современным операционным системам

К основным требованиям, предъявляемым к операционной системе, относится выполнение функций управления ресурсами и обеспечение удобного интерфейса для пользователей и прикладных программ. Современная ОС должна поддерживать мультипрограммную обработку, виртуальную память, свопинг, многооконный графический интерфейс пользователя, а также соблюдать следующие эксплуатационные требования:

– Расширяемость: ОС должна быть способна принимать дополнения и изменения без нарушения целостности системы, благодаря чему её полезная жизнь может измеряться десятилетиями.

– Переносимость: Код ОС должен легко переноситься с одного процессора на другой и с одной аппаратной платформы на другую, что может включать различные архитектуры.

– Совместимость: ОС должна обладать средствами выполнения прикладных программ, написанных для других операционных систем, чтобы обеспечить совместимость с ними.

– Надежность и отказоустойчивость: Система должна быть защищена от внутренних и внешних ошибок, сбоев и отказов. Ее действия должны быть предсказуемыми, а приложения не должны иметь возможности наносить вред ОС.

– Безопасность: Современная ОС должна защищать данные и ресурсы вычислительной системы от несанкционированного доступа.

– Производительность: ОС должна демонстрировать хорошую скорость работы и время реакции, максимально соответствующее возможностям аппаратной платформы.

Тема 1.2. Назначение и основные функции операционных систем

Основная задача операционной системы (ОС) заключается в предоставлении пользователю или программисту расширенной виртуальной машины, которая упрощает процесс программирования и взаимодействия с аппаратным обеспечением реального компьютера или сети. Операционная система рассматривается как комплекс взаимосвязанных программ, выполняющих роль интерфейса между приложениями и пользователями с одной стороны и аппаратурой компьютера с другой. Таким образом, ОС выполняет две ключевые *группы функций:*

1. Предоставление пользователю или программисту виртуальной машины вместо реального аппаратного обеспечения.

2. Оптимизация использования компьютера с целью наиболее рационального управления его ресурсами в соответствии с определенными критериями.

Функции ОС по управлению ресурсами компьютера

Управление ресурсами вычислительной системы для обеспечения их наиболее эффективного использования является основным назначением любой операционной системы. К основным ресурсам современных вычислительных систем относятся процессоры и оперативная память. Эти ресурсы распределяются между процессами, где процесс (или задача) представляет собой программу в стадии выполнения, а программа – статический объект, в то время как процесс – динамический, который создается при запуске программы.

Основные критерии эффективности, по которым ОС организует управление ресурсами, включают пропускную способность вычислительной системы и время реакции. Управление ресурсами включает решения следующих общих *задач:*

– Планирование ресурса: определение, какому процессу, когда и в каком количестве следует выделить данный ресурс.

– Удовлетворение запросов на ресурсы .

– Отслеживание состояния и учет использования ресурса: поддержание актуальной информации о состоянии ресурса (занят или свободен) и его распределении.

– Разрешение конфликтов между процессами.

Большинство этих функций осуществляется операционной системой автоматически и остаются недоступными для программиста.

Управление процессами

Подсистема управления процессами распределяет процессорное время между несколькими одновременно существующими в системе процессами. Она занимается созданием и уничтожением процессов, предоставляет не-

обходимые системные ресурсы, поддерживает синхронизацию процессов и обеспечивает их взаимодействие.

Для каждого вновь создаваемого процесса ОС генерирует информационные структуры, содержащие данные о потребностях процесса в ресурсах, а также фактически выделенные ему ресурсы. Чтобы процесс мог выполняться, операционная система должна выделить ему область оперативной памяти для размещения кода и данных, а также предоставить необходимое количество процессорного времени.

В мультипрограммной операционной системе может одновременно существовать несколько процессов. Некоторые из них инициируются пользователями или приложениями (пользовательские процессы), в то время как другие, называемые системными, создаются самой операционной системой для выполнения ее функций.

Среди важнейших *задач* ОС – защита ресурсов, выделенных каждому процессу, от доступа других процессов. Одним из наиболее защищаемых ресурсов являются области оперативной памяти, в которых хранятся коды и данные процесса. Совокупность всех этих областей называется адресным пространством процесса.

На протяжении своего существования процесс может многократно прерываться и возобновляться. Для восстановления выполнения необходимо восстановить состояние его контекста. Контекст процесса включает сведения о состоянии операционной среды, таких как регистры, программный счетчик, режим работы процессора, указатели на открытые файлы, информация о незавершенных операциях ввода-вывода, коды ошибок системных вызовов и другие данные.

Управление памятью

Функции операционной системы (ОС) по управлению памятью включают:

– Отслеживание свободной и занятой памяти.

– Выделение памяти процессам и освобождение ее по завершении процессов.

– Защита памяти.

– Вытеснение процессов из оперативной памяти на диск в случае нехватки места и возвращение их обратно при освобождении пространства.

– Настройка адресов программы на соответствующую область физической памяти.

Управление памятью подразумевает распределение доступной физической памяти между всеми активными процессами в системе, загрузку кодов и данных в выделенные области памяти, настройку адресно-зависимых частей кода на физические адреса выделенной памяти, а также защиту этих областей для каждого процесса.

Одним из популярных методов управления памятью в современных операционных системах является механизм виртуальной памяти. Он позволяет хранить все данные, необходимые программе, на диске и отображать их в физическую память частями (секциями или страницами) по мере необходимости. Это создает для программиста иллюзию работы с однородной оперативной памятью большой емкости, значительно превышающей реальный объем физической памяти. При перемещении кода и данных между оперативной памятью и диском подсистема виртуальной памяти выполняет трансляцию виртуальных адресов, полученных в результате компиляции и линковки программы, в физические адреса ячеек оперативной памяти. Важно, что все операции по перемещению данных между оперативной памятью и диском, а также трансляция адресов выполняются ОС незаметно для программиста.

Защита памяти

Защита памяти представляет собой механизм, который предотвращает возможность записи или чтения памяти одной задачей, назначенной другой. Реализованные в ОС средства защиты обеспечивают недоступность чужих областей памяти для процессов, защищая систему от несанкционированного доступа.

Управление файлами и внешними устройствами

Файл представляет собой простую неструктурированную последовательность байтов с символьным именем. Для удобства работы с данными файлы группируются в каталоги, которые, в свою очередь, могут образовывать более высокоуровневые группы. Пользователь может выполнять различные действия с файлами и каталогами через ОС, такие как поиск, удаление, вывод содержимого на внешние устройства (например, на экран), а также изменение и сохранение данных.

Файловая система ОС преобразует символьные имена файлов, с которыми работает пользователь или программист, в физические адреса данных на диске. Она организует совместный доступ к файлам и защищает их от несанкционированного доступа.

При выполнение своих задач файловая система тесно взаимодействует с подсистемой управления внешними устройствами, которая отвечает за передачу данных между дисками и оперативной памятью по запросам файловой системы. Эта подсистема, также известная как подсистема ввода-вывода, выступает в роли интерфейса для всех устройств, подключенных к компьютеру. Программа, управляющая конкретной моделью устройства, называется драйвером (от английского «drive» – управлять). Наличие разнообразных драйверов в ОС гарантирует возможность подключения большого числа внешних устройств различных производителей.

Хотя прикладные программисты могут использовать интерфейсы драйверов для разработки своих программ, это может быть не удобно, так как такие интерфейсы часто подразумевают низкоуровневые операции с множеством деталей. Поддержание высокоуровневого унифицированного интерфейса прикладного программирования к разнообразным устройствам ввода-вывода является одной из ключевых задач операционной системы.

Поддержка пользовательского интерфейса

Операционная система обязана предоставлять удобный интерфейс как для прикладных программ, так и для пользователей, работающих за компьютером. Современные ОС поддерживают множество функций пользовательского интерфейса, которые способствуют интерактивной работе, включая алфавитно-цифровые терминалы, графические интерфейсы, модули голосовых команд, речевое управление и интерфейсы, основанные на анализе биологических сигналов.

При использовании алфавитно-цифрового терминала пользователь имеет доступ к системе команд, отражающей функциональные возможности ОС. Команды могут вводиться как в интерактивном режиме, так и считываться из специального командного файла, содержащего последовательность команд. Программный модуль, который отвечает за чтение команд из этого файла, часто называется командным интерпретатором.

Ввод команд может быть упростен при поддержке графического пользовательского интерфейса (GUI), где пользователь выбирает необходимые действия с помощью мыши, щелкая на пункты меню или графические значки.

Защита данных и поддержка администрирования

Безопасность данных в вычислительной системе осуществляется средствами отказоустойчивости ОС, направленными на защиту от сбоев аппаратуры, ошибок программного обеспечения и несанкционированного доступа. Функции защиты ОС тесно связаны с администрированием, поскольку именно администратор определяет права пользователей на доступ к различным ресурсам системы – файлам, каталогам, принтерам, сканерам и др. Кроме того, администратор ограничивает возможности пользователей в выполнении определенных системных действий.

Функции аудита операционной системы, фиксирующие все события, влияющие на безопасность системы, являются важным инструментом защиты данных. Список событий для отслеживания формируется администратором.

Отказоустойчивость, как правило, реализуется через резервирование, при этом ОС поддерживает несколько копий данных на различных дисках или накопителях. Также резервируются принтеры и другие устройства ввода-вывода. В случае отказа одного из избыточных устройств ОС должна оперативно и прозрачно для пользователя провести реконфигурацию системы и продолжить работу с резервным устройством. Особым случаем от-

казоустойчивости является мультипроцессирование, когда система может продолжать работу при выходе из строя одного из процессоров, хотя и с уменьшенной производительностью.

Поддержка интерфейса прикладного программирования

Возможности ОС доступны прикладным программистам через набор функций, известный как интерфейс прикладного программирования (Application Programming Interface, API). Особенности конкретной операционной системы для разработчиков приложений проявляются через возможности API. Это позволяет операционным системам с различной внутренней организацией, но с одинаковым набором функций API, выглядеть одинаково, что упрощает стандартизацию и обеспечивает переносимость приложений между разными ОС, соответствующими общему стандарту API.

Прикладные программы осуществляют обращения к функциям API через системные вызовы. Процесс вызова функций ОС схож с вызовом подпрограммы: информация, необходимая для ОС – идентификатор команды и данные – помещается в определенные области памяти, регистры или стек. Затем управление передается операционной системе, которая выполняет требуемую операцию и возвращает результат через память, регистры или стек. Если операция завершается с ошибкой, результат содержит информацию о возникшей ошибке.

Способ реализации системных вызовов зависит от структурной организации ОС и особенностей аппаратной платформы, а также от используемого языка программирования.

Тема 1.3. Архитектурные особенности операционных систем

Обобщённая структура операционной системы

Архитектура операционной системы (ОС) относится к её структурной организации, состоящей из различных программных модулей. Обычно ОС включает исполняемые и объектные модули стандартных форматов, библиотеку различных типов, модули исходного текста программ, специализированные программные модули (например, загрузчики ОС и драйверы ввода-вывода), конфигурационные файлы, документацию, справочные системы и другое.

Наиболее распространённый подход к структурированию ОС заключается в разделении всех её модулей на две основные группы:

Ядро – модули, выполняющие основные функции ОС.

Вспомогательные модули – модули, обеспечивающие дополнительные функции.

Ядро управляет базовыми задачами ОС, такими как управление процессами, памятью и устройствами ввода-вывода. Оно является центральной частью операционной системы, без которой ОС не сможет выполнять ни одну из своих функций. В состав ядра входят функции, решающие внутрисистемные задачи, такие как переключение контекстов, загрузка и выгрузка страниц, обработка прерываний. Эти функции недоступны для приложений. Ядро также поддерживает приложения, создавая для них прикладную программную среду. Приложения могут обращаться к ядру через системные вызовы для выполнения различных действий, таких как открытие и чтение файлов, вывод информации на дисплей, получение системного времени и т. д. Функции ядра, доступные приложениям, формируют интерфейс прикладного программирования (API).

Для обеспечения высокой скорости работы ОС большинство модулей ядра остаются в оперативной памяти, то есть являются резидентными.

Обычно ядро представляется в виде специального программного модуля, формат которого отличается от формата пользовательских приложений. В разных ОС термин «ядро» может иметь различные трактовки. Одним из ключевых свойств ядра является работа в привилегированном режиме.

Вспомогательные модули ОС выполняют менее критичные функции, такие как программы для архивирования данных, дефрагментации диска, текстовые редакторы и др. Эти модули могут быть представлены в виде приложений или библиотек процедур.

Вспомогательные модули обычно подразделяются на несколько групп:

Утилиты – программы, решающие конкретные задачи управления и обслуживания компьютерной системы, например, программы для сжатия дисков или архивирования данных на магнитную ленту.

Системные обрабатывающие программы – текстовые или графические редакторы, компиляторы, компоновщики, отладчики.

Программы дополнительного сервиса – специальные варианты пользовательских интерфейсов, калькуляторы, игры и т. д.

Библиотеки процедур – различные библиотеки, которые упрощают разработку приложений, такие как библиотеки математических функций или функций ввода-вывода.

Разделение операционной системы на ядро и вспомогательные модули обеспечивает её лёгкую расширяемость. Для добавления новой высокоуровневой функции достаточно разработать новое приложение, не внося изменений в ядро системы.

Одним из важных аспектов архитектуры ОС, основанной на ядре, является возможность защиты кодов и данных операционной системы, благодаря выполнению функций ядра в привилегированном режиме.

Архитектура операционных систем на основе монолитного ядра:

Монолитное ядро представляет собой набор процедур, каждая из которых может вызывать любую другую. Все эти процедуры функционируют в привилегированном режиме, что делает монолитное ядро архитектурой, при которой все компоненты ОС являются частями единой программы, используют общие структуры данных и взаимодействуют посредством непосредственных вызовов процедур. В данном случае ядро совпадает с операционной системой целиком.

Во многих операционных системах с монолитным ядром процесс сборки ядра, или его компиляция, выполняется отдельно для каждого компьютера, на который устанавливается ОС. При этом имеется возможность выбрать список оборудования и программных протоколов, поддержка которых будет интегрирована в ядро. Типичный набор аппаратной поддержки ОС включает:

– Средства поддержки привилегированного режима;
– Средства трансляции адресов;
– Средства переключения процессов;
– Система прерываний;
– Системный таймер;
– Средства защиты областей памяти.

Поддержка привилегированного режима обычно основывается на системном регистре процессора, известном как «слово состояния» машины или процессора. Этот регистр содержит признаки, определяющие режимы работы процессора, включая текущий режим привилегий. Смена режима привилегий происходит за счёт изменения слова состояния машины в результате прерывания или выполнения привилегированной команды. Средства поддержки привилегированного режима проверяют допустимость выполнения инструкций процессора активной программой в зависимости от текущего уровня привилегий.

Средства трансляции адресов отвечают за преобразование виртуальных адресов, содержащихся в кодах процесса, в адреса физической памяти. Таблицы, используемые для трансляции адресов, часто занимают большой объем, поэтому хранятся в оперативной памяти, а аппаратное обеспечение процессора содержит лишь указатели на эти области.

Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста активного процесса. Контекст обычно включает данные всех регистров общего назначения, регистр флагов операций (таких как нулевой флаг, флаг переноса, флаг переполнения и др.), а также системные регистры и указатели, связанные с конкретным процессом, например, указатель на таблицу трансляции адресов процесса. Контексты приостановленных про-

цессов хранятся в оперативной памяти, доступ к которым осуществляется с помощью указателей процессора.

Система прерываний обеспечивает компьютеру возможность реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, а также быстро переключаться между различными программами. Механизм прерываний необходим для оповещения процессора о возникновении непредсказуемых событий или событий, не синхронизированных с рабочим циклом процессора.

Системный таймер, часто реализованный в виде быстродействующего регистра-счетчика, нужен ОС для обеспечения интервальных периодов времени.

Средства защиты областей памяти предоставляют аппаратные механизмы проверки прав на выполнение операций с данными, находящимися в определённых областях памяти, таких как чтение, запись или выполнение.

Поскольку ядро функционирует как единая программа, его перекомпиляция является единственным способом добавления новых компонентов или исключения неиспользуемых.

Сервисные процедуры выполняются в привилегированном режиме, в то время как пользовательские программы действуют в непривилегированном режиме. Для перехода между уровнями привилегий может использоваться главная сервисная программа, определяющая тип системного вызова, проверяющая корректность входных данных и передающая управление соответствующей сервисной процедуре с переключением в привилегированный режим.

Достоинство монолитного ядра заключается в скорости вызова системных функций, тогда как

Недостаток заключается в низкой надёжности и отказоустойчивости.

Особенности работы ядра в привилегированном режиме

Для функционирования ядра операционной системы (ОС) в привилегированном режиме, аппаратное обеспечение компьютера должно поддерживать по меньшей мере два режима работы: пользовательский режим (user mode) и привилегированный режим, также известный как режим ядра (kernel mode) или режим супервизора (supervisor mode). В этом контексте предполагается, что операционная система или её части работают в привилегированном режиме, тогда как приложения функционируют в пользовательском режиме.

Приложения находятся в подчинённом положении благодаря запрету выполнения критически важных команд в пользовательском режиме, относящихся к переключению процессов, управлению устройствами ввода-вывода и доступу к механизмам распределения и защиты памяти.

Разрешение на выполнение инструкции доступа к памяти для приложения предоставляется только в том случае, если инструкция обращается к памяти, выделенной этому приложению операционной системой, а доступ запрещается при обращении к участкам памяти, занятым ОС или другими приложениями. Каждое приложение работает в своём собственном адресном пространстве, что позволяет локализовать проблемы приложения в его области памяти и предотвращает их влияние на другие приложения и саму ОС.

Приложения в пользовательском режиме защищены от вмешательства со стороны других приложений за счет того, что код ядра, выполняющийся в привилегированном режиме, имеет доступ ко всем областям памяти, но сам по себе является защищённым. Запросы приложений к ядру для выполнения системных функций осуществляются через системные вызовы.

Системный вызов инициирует переключение процессора из пользовательского режима в привилегированный (рис. 1), а возвращение к приложению – переключение обратно в пользовательский режим. Этот процесс работает медленнее, чем вызов процедуры без смены режима, из-за присутствия двух этапов переключения.

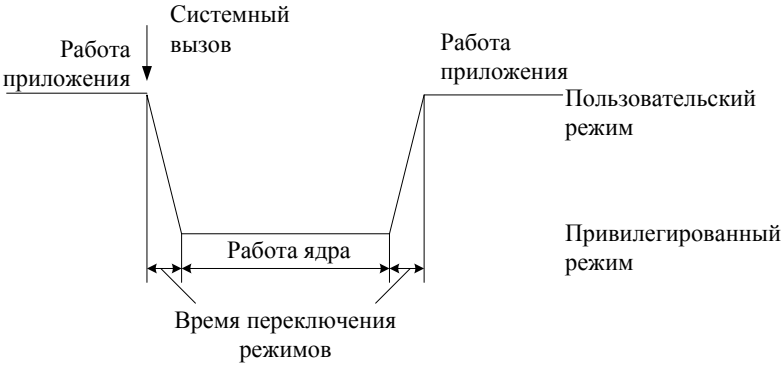


Рис. 1. Смена режимов при выполнении системного вызова к привилегированному ядру

Многослойная структура операционной системы

В вычислительных системах под управлением ОС, основанной на ядре, можно выделить три иерархически расположенных слоя: нижний слой представляет собой аппаратное обеспечение, промежуточный слой – ядро, а верхний слой включает утилиты, обрабатывающие программы и приложения.

Многослойный подход является универсальным и эффективным методом декомпозиции сложных систем, включая программные. Каждый слой обслуживает вышележащий слой, выполняя определённые функции,

которые образуют межслойный интерфейс. С помощью функций нижележащего слоя следующий слой определяет свои более сложные и мощные функции, которые становятся основой для создания ещё более мощных функций вышележащего слоя. Строгие правила касаются только взаимодействия между слоями, тогда как модули внутри одного слоя могут взаимодействовать произвольно. *Каждый модуль может выполнять свои задачи самостоятельно или обращаться за помощью к другим модулям своего слоя или к нижележащему слою через межслойный интерфейс.*

Такая организация значительно упрощает разработку системы, поскольку позволяет сначала определить функции слоёв и межслойные интерфейсы «сверху вниз», а затем постепенно наращивать мощность функций «снизу вверх». При модернизации системы можно изменять модули внутри слоя, при этом не затрагивая остальные слои, если сохраняются межслойные интерфейсы.

Ядро может состоять из следующих слоёв

1. *Средства аппаратной поддержки ОС:* средства поддержки привилегированного режима, система прерываний, средства переключения контекстов процессов, средства защиты областей памяти и другие.

2. *Машинно-зависимые компоненты ОС:* программные модули, отражающие специфику аппаратной платформы компьютера.

3. *Базовые механизмы ядра:* выполнение примитивных операций ядра, таких как программное переключение контекстов процессов, диспетчеризация прерываний, перемещение страниц из памяти на диск и обратно.

4. *Менеджеры ресурсов:* функциональные модули для управления основными ресурсами вычислительной системы, включая диспетчеров процессов, ввода-вывода, файловой системы и оперативной памяти.

5. *Интерфейс системных вызовов:* верхний слой ядра, который взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс операционной системы.

Выбор количества слоев ядра

Определение количества слоев в ядре операционной системы представляет собой баланс между производительностью и расширяемостью. Увеличение числа слоев может привести к замедлению работы ядра из-за дополнительных накладных расходов, связанных с межслойным взаимодействием. В то же время уменьшение числа слоев может ухудшить расширяемость и логическую структуру системы.

Микроядерная архитектура

Основная концепция микроядерной архитектуры заключается в следующем: в привилегированном режиме функционирует только малый объем операционной системы, который называется микроядром. Это микроядро защищено от остальных компонентов ОС и приложений.

Обычно микроядро включает машинно-зависимые модули, а также модули, отвечающие за базовые функции ядра, такие как управление процессами, обработка прерываний, управление виртуальной памятью, пересылка сообщений и управление устройствами ввода-вывода, включая загрузку или чтение регистров устройств.

Все остальные более высокоуровневые функции ядра реализуются в виде приложений, функционирующих в пользовательском режиме. Менеджеры ресурсов, перенесенные в пользовательский режим, называют серверами ОС. Они предназначены для обслуживания запросов локальных приложений и других модулей операционной системы.

Обращение к функциям ОС, оформленным как серверы, осуществляется следующим образом: клиент (это может быть как прикладная программа, так и другой компонент ОС) отправляет запрос на выполнение определенной функции соответствующему серверу в виде сообщения. Непосредственная передача сообщений между приложениями невозможна из-за изоляции их адресных пространств. Однако микроядро, работающее в привилегированном режиме, имеет доступ ко всем адресным пространствам и может выполнять роль посредника. Микроядро передает сообщение с именем и параметрами вызываемой процедуры нужному серверу, который затем выполняет запрошенную операцию. После этого ядро возвращает результаты клиенту через другое сообщение. Таким образом, работа микроядерной операционной системы строится на модели клиент-сервер, где микроядро выступает в роли транспортного механизма (рис. 2).

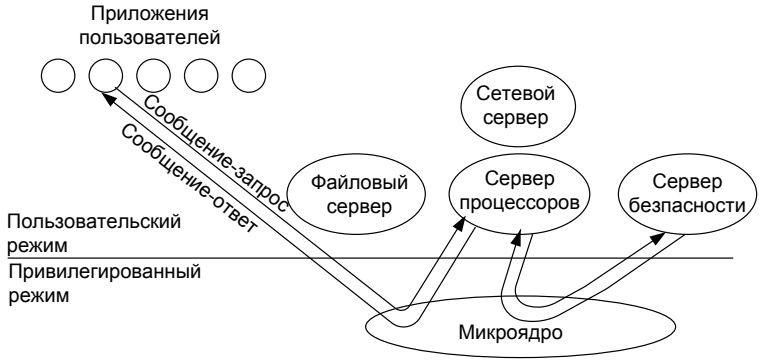


Рис. 2. Реализация системного вызова в микроядерной архитектуре

Преимущества микроядерной архитектуры

Переносимость: Высокая степень переносимости обеспечивается тем, что весь машинно-зависимый код сосредоточен в микроядре. Это позволя-

ет минимизировать количество изменений при переносе системы на новый процессор, и все изменения логически сгруппированы.

Расширяемость: Микроядерная архитектура обладает высокой степенью расширяемости. Добавление новой подсистемы сводится к разработке нового приложения, не затрагивая целостность микроядра. Это также позволяет эффективно уменьшать общее количество компонентов ОС.

Надежность: Использование микроядерной модели повышает надежность операционной системы, так как каждый сервер выполняется как отдельный процесс с собственной областью памяти, что защищает его от влияния других серверов. Если один из серверов выходит из строя, он может быть перезапущен, не влияя на работу остальных.

Поддержка распределенных вычислений: Микроядро эффективно поддерживает распределенные вычисления, используя механизмы, аналогичные сетевым, для взаимодействия клиентов и серверов через обмен сообщениями. Серверы микроядерной ОС могут функционировать как на одном, так и на разных компьютерах, переход к распределенной обработке требует минимальных изменений – лишь замены локального транспорта на сетевой.

Недостатки микроядерной архитектуры

Одним из главных недостатков микроядерной архитектуры является снижение производительности. Операционная система, основанная на микроядре, при равных условиях будет всегда менее производительной по сравнению с классической архитектурой. Это обусловлено тем, что в классической операционной системе выполнение системного вызова требует двух переключений режимов, тогда как при микроядерной архитектуре этот процесс включает, как минимум, четыре переключения. Такое количество дополнительных переключений требует значительно большего числа ресурсов процессора и повышает время обработки.

Тема 1.4. Основные понятия, концепции ОС

Основные понятия, концепции ОС. Программное прерывание, аппаратное прерывание, исключительная ситуация, файлы

Основные понятия, концепции ОС

Системные вызовы (system calls) представляют собой интерфейс между операционной системой и пользовательскими программами. Они позволяют создавать, удалять и управлять различными объектами, такими как процессы и файлы. Пользовательская программа инициирует системный вызов для запроса сервиса у операционной системы. Имеются

специализированные библиотеки процедур, которые загружают значения в машинные регистры и инициируют прерывание процессора, после чего управление передаётся обработчику вызова, входящему в ядро ОС. Эти библиотеки предназначены для того, чтобы сделать системный вызов схожим с обычным вызовом подпрограммы.

Основное отличие заключается в том, что при системном вызове происходит переход в привилегированный режим или режим ядра (kernel mode). Поэтому системные вызовы также иногда называют программными прерываниями, в отличие от аппаратных прерываний, которые обычно называются просто прерываниями.

В привилегированном режиме исполняется код ядра ОС, который выполняется в адресном пространстве и контексте вызвавшего его процесса. Это обеспечивает ядру полный доступ к памяти пользовательской программы, что позволяет просто передавать адреса одной или нескольких областей памяти для параметров вызова и для результатов.

В большинстве операционных систем системный вызов инициируется командой программного прерывания (INT), что делает программное прерывание синхронным событием.

Прерывания

Прерывание (hardware interrupt) представляет собой событие, генерируемое внешним устройством (по отношению к процессору). Аппаратные прерывания позволяют устройству информировать центральный процессор о возникновении события, требующего немедленного реагирования (например, нажата клавиша пользователем), или сообщить о завершении асинхронной операции ввода-вывода (например, завершено чтение данных с диска в оперативную память).

Аппаратное прерывание является асинхронным событием, то есть оно происходит независимо от текущего состояния процесса, выполняемого процессором, и обработка данного прерывания не зависит от того, какой процесс в данный момент является активным.

Исключительные ситуации

Исключительная ситуация (exception) – это событие, возникающее при попытке выполнения недопустимой команды, доступа к ресурсу без достаточных привилегий или обращения к отсутствующей странице памяти. Исключительные ситуации, подобно системным вызовам, являются синхронными событиями и происходят в контексте текущей задачи.

Исключительные ситуации делятся на исправимые и неисправимые. *Исправимые* ситуации, например отсутствие необходимой информации в оперативной памяти, могут быть устранены, после чего программа продолжит выполнение. Возникновение исправимых исключительных ситуаций в процессе работы ОС считается нормальным явлением.

Неисправимые исключительные ситуации, как правило, возникают в результате ошибок в программном обеспечении. Ответ операционной системы на такие исключительные ситуации обычно заключается в завершении программы, вызвавшей ошибку.

РАЗДЕЛ 2. ПРОЦЕССЫ

Тема 2.1. Понятие процесса

Одним из ключевых элементов операционной системы, непосредственно воздействующим на работу вычислительной машины, является подсистема управления процессами. Процесс в контексте ОС можно рассматривать как единицу работы, которая делает заявку на использование системных ресурсов.

Примеры процессов могут включать прикладные программы пользователей, различные утилиты и другие обрабатываемые системы. Процессом может быть, например, редактирование текста, трансляция исходного кода, компоновка и его исполнение. При этом трансляция первого исходного кода считается одним процессом, а трансляция второго – другим, так как транслятор, действующий как общая программа, работает с различными данными.

Понятие процесса

Процесс (или задача) – это программа, находящаяся на стадии выполнения. Программа, с другой стороны, представляет собой статический объект, состоящий из файла с кодом и данными. Процесс – это динамический объект, который появляется в операционной системе, когда пользователь или сама система решает «запустить программу», тем самым создавая новую единицу вычислительной работы.

Подсистема управления процессами

Подсистема управления процессами ответственна за планирование выполнения процессов. Она распределяет процессорное время между несколькими одновременно работающими процессами, создает и уничтожает процессы, обеспечивает их ресурсами, поддерживает синхронизацию и взаимодействие между ними.

Чтобы процесс мог быть выполнен, операционная система должна выделить ему область оперативной памяти для размещения кода и данных, а также предоставить необходимое количество процессорного времени.

Основные функции подсистемы управления процессами:

1. Распределение процессорного времени между несколькими одновременно работающими процессами.
2. Создание и уничтожение процессов.
3. Обеспечение процессов необходимыми системными ресурсами.
4. Поддержка взаимодействия между процессами.

Обычно процесс включает два основных компонента:

1.Объект ядра: элемент, через который ОС управляет процессом и хранит статистическую информацию о нем.

2. Адресное пространство: область, в которой находятся код и данные всех модулей процесса.

Процессы являются инертными; чтобы процесс мог выполнить какую-либо задачу, в нем необходимо создать поток. Потоки отвечают за выполнение кода, находящегося в адресном пространстве процесса. Каждый процесс может иметь несколько потоков, которые одновременно исполняют код.

При создании процесса первичный поток часто создается автоматически системой. Для каждого нового процесса ОС генерирует информационные структуры, содержащие данные о потребностях процесса в ресурсах вычислительной системы и фактическом выделении ресурсов.

Состояния процесса (потока)

В многозадачной системе выделяются три основных состояния процесса (рис. 3):

1. *Выполнение:* активное состояние, когда процесс обладает всеми необходимыми ресурсами и исполняется процессором.

2. *Ожидание:* пассивное состояние, при котором процесс заблокирован, ожидает события, например, завершения операции ввода-вывода или получения сообщения от другого процесса.

3. *Готовность:* также пассивное состояние, в котором процесс готов к выполнению, имеет все необходимые ресурсы, но процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования, реализуемым в данной операционной системе.

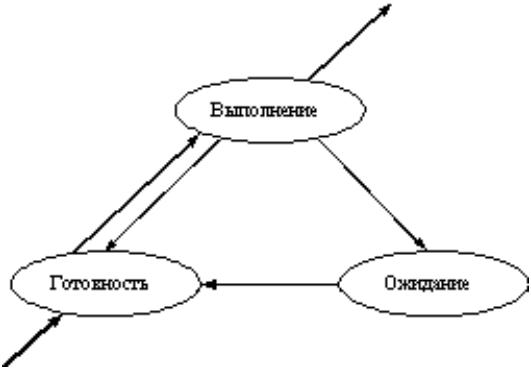


Рис. 3. Граф состояний процесса в многозадачной среде

В однопроцессорной системе только один процесс может находиться в состоянии ВЫПОЛНЕНИЕ, тогда как в состояниях ОЖИДАНИЕ и ГОТОВНОСТЬ могут быть несколько процессов. Процессы образуют очереди ожидающих и готовых процессов соответственно.

Жизненный цикл процесса начинается в состоянии ГОТОВНОСТЬ, когда процесс готов к выполнению и ожидает своей очереди. При активации он переходит в состояние ВЫПОЛНЕНИЕ, где остается до тех пор, пока не освободит процессор, перейдя в состояние ОЖИДАНИЯ какого-либо события, или не будет вытеснен, например, из-за исчерпания своего квота процессорного времени. В последнем случае процесс возвращается в состояние ГОТОВНОСТЬ. Процесс также переходит в состояние Готовность из состояния ОЖИДАНИЯ после наступления ожидаемого события.

Контекст и дескриптор процесса (потока)

В процессе существования выполнение задачи может многократно прерываться и возобновляться. Для того, чтобы продолжить выполнение процесса, необходимо восстановить состояние его операционной среды. Это состояние включает информацию о значениях регистров, режиме работы процессора, указатели на открытые файлы, сведения о незавершенных операциях ввода-вывода, коды ошибок, системные вызовы, выполняемые данным процессом, и другие данные, которые характеризуют состояние вычислительной среды в момент прерывания. Данная информация называется контекстом процесса и содержит все сведения, нужные для возобновления выполнения процесса с места прерывания.

В дополнение к этому, для реализации планирования процессов операционной системе требуется вспомогательная информация: идентификатор процесса, его текущее состояние, данные о степени привилегированности, местоположение кодового сегмента и другую важную информацию. В некоторых операционных системах, таких как UNIX, такую информацию называют дескриптором процесса. Дескриптор процесса содержит данные, необходимые ядру на протяжении всего жизненного цикла процесса, независимо от его состояния – активного или пассивного, а также независимо от того, загружен ли образ процесса в оперативную память или выгружен на диск.

В отличие от контекста, дескриптор процесса включает более актуальную информацию, которая должна быть доступна подсистеме планирования процессов. Контекст, в свою очередь, содержит менее актуальные сведения и используется операционной системой только после принятия решения о восстановлении прерванного процесса.

Очереди процессов состоят из дескрипторов отдельных процессов, объединенных в списки. Каждый дескриптор, кроме прочего, содержит хотя бы один указатель на соседний дескриптор в очереди. Такая структура

позволяет легко переупорядочивать очереди, добавлять и исключать процессы, а также переводить их из одного состояния в другое.

Тема 2.2. Блок управления процессом. Одноразовые операции

Операции над процессами и связанные с ними понятия. Набор операций

Процесс не может самостоятельно изменять свои состояния. Эта задача возложена на операционную систему, которая осуществляет операции по изменению состояния процессов. Количество таких операций соответствует числу стрелок на диаграмме состояний процессов. Удобно сгруппировать их в три основные пары:

– ***Создание процесса*** – Завершение процесса;

– ***Приостановка процесса*** (переход из состояния «исполнение» в «готовность»);

– ***Запуск процесса*** (переход из состояния «готовность» в «исполнение»);

– ***Блокирование процесса*** (переход из состояния «исполнение» в «ожидание»);

– ***Разблокирование процесса*** (переход из состояния «ожидание» в «готовность»).

Операции создания и завершения процесса являются одноразовыми, так как их применяют к процессу не более одного раза (некоторые системные процессы могут оставаться активными на протяжении всего времени работы вычислительной системы). Все остальные операции, связанные с изменением состояния процессов, такие как запуск или блокировка, обычно являются многоразовыми. Рассмотрим подробнее, как операционная система выполняет данные операции.

Одноразовые операции

Жизненный цикл процесса в компьютере начинается с его создания. Каждая операционная система, оперирующая концепцией процессов, должна иметь средства для их генерации. В простейших системах (например, в тех, что предназначены для запуска лишь одного конкретного приложения) все процессы могут быть инициированы на этапе старта системы. Более сложные операционные системы создают процессы динамически, по мере необходимости.

Инициатором создания нового процесса может выступать либо пользовательский процесс, который осуществляет специальный системный вызов, либо сама операционная система, также являясь процессом. Про-

цесс, создавший новый процесс, называется родительским (parent process), а вновь созданный – дочерним (child process). Дочерние процессы могут, в свою очередь, создавать другие дочерние процессы, образуя генеалогическую структуру, известную как генеалогический лес (рис. 4). Все пользовательские процессы и некоторые процессы ОС обычно принадлежат одному дереву этого леса. В ряде систем такой лес может упроститься до одного дерева.

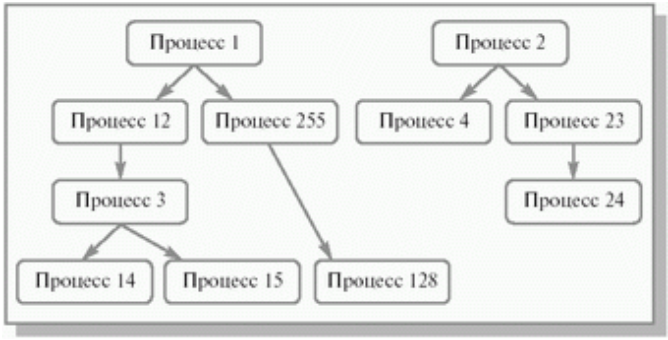


Рис. 4. Упрощенный генеалогический лес процессов.
Стрелочка означает отношение родитель–ребенок

При создании процесса операционная система выделяет новый блок управления процессом с состоянием «рождение» и начинает заполнять его. Новый процесс получает уникальный идентификационный номер. Поскольку для хранения этого номера в ОС используется ограниченное число битов, количество одновременно работающих процессов должно быть ограничено. После завершения какого-либо процесса его идентификатор может быть повторно использован.

Обычно дочернему процессу требуются определенные ресурсы: память, файлы, устройства ввода-вывода и т. д. Существуют два подхода к выделению ресурсов. Новый процесс может получить часть ресурсов родителя, возможно разделяя их с другими дочерними процессами, или же получить свое ресурсы непосредственно от операционной системы. Информация о выделенных ресурсах записывается в блок управления процессом.

После заполнения блока управления процессом оставшейся информацией состояние нового процесса изменяется на «готовность». Родительский процесс может продолжать выполнение одновременно с дочерними процессами или ожидать их завершения.

После завершения работы процесса операционная система переводит его в состояние «завершение» и освобождает все ассоциированные с ним ресурсы, делая необходимые записи в блоке управления процессом. Сам

блок управления процессом не уничтожается сразу, а остается в системе на некоторое время, чтобы родительский процесс мог запросить информацию о причине завершения дочернего процесса и/или статистику его работы. Эта информация сохраняется в РСВ (блоке управления процессом) завершившего работу процесса до обращения родительского процесса или до завершения его жизненного цикла. После этого все следы завершившегося процесса исчезают из системы.

Тема 2.3. Многоразовые операции

Многоразовые операции

В процессе работы операционной системы один и тот же процесс может многократно выполнять свои задачи. Это происходит по нескольким причинам: обращению процессов к операционной системе с запросами на ресурсы, выполнению системных функций, взаимодействию с другими процессами, а также реагированию на прерывания от таймеров и устройств ввода/вывода. Основные *состояния процесса* можно разделить на два:

– *Активное состояние:* процесс участвует в конкуренции за ресурсы компьютера.

– *Пассивное состояние:* процесс известен операционной системе, но не осуществляет активных действий в отношении ресурсов.

Существуют определённые правила перехода между этапами выполнения процессов, которые могут зависеть от системных событий или действий пользователей. При выполнении многоразовых операций над процессами операционная система выполняет следующие *действия.*

1. *Запуск процесса:* Из процессов, находящихся в состоянии «готовность», операционная система выбирает один для исполнения. Операционная система должна обеспечить наличие в оперативной памяти всей информации, необходимой для выполнения этого процесса. Следующим шагом состояние процесса изменяется на «исполнение», восстанавливаются значения регистров, и управление передается команде, на которую указывает счетчик команд. Все данные, необходимые для восстановления контекста, извлекаются из блока управления процессом (БУП).

2. *Приостановка процесса:* Процесс может быть приостановлен из состояния «исполнение» по различным причинам:

– Супервизор операционной системы переводит процесс в состояние «готовность» из-за возникновения более приоритетной задачи или истечения времени выполнения.

– Процесс блокируется из-за отсутствия необходимого ресурса или ожидает ввода/вывода, а также по команде оператора.

После приостановки процесса, процессор сохраняет счетчик команд и регистры в стеке исполняемого процесса и передает управление в специальную область для обработки прерывания. В этой области, как правило, находится часть операционной системы, которая сохраняет динамическую часть контекста процесса, переводит его в состояние «готовность» и начинает обработку прерывания.

3. *Блокирование процесса:* Процесс блокируется, если он не может продолжать работу, ожидая определённого события в системе. Для этого он обращается к операционной системе через специальный системный вызов. Операционная система обрабатывает этот вызов, при необходимости сохраняет часть контекста процесса в его PCB и переводит процесс из состояния «исполнение» в «ожидание».

4. *Разблокирование процесса:* Когда происходит определенное событие и оно определяется операционной системой, происходит проверка на наличие процесса в состоянии «ожидание». Если такой процесс найден, операционная система переводит его в состояние «готовность» и выполняет действия, связанные с произошедшим событием, например, инициализацию операции ввода-вывода для ожидающего процесса.

Процесс может перейти из состояния «блокирование» в «готовность» в следующих случаях:

– По команде пользователя;

– При выборе процесса супервизором;

– По вызову из другой задачи, когда один процесс может создавать, инициировать, приостанавливать, останавливать или уничтожать другой процесс;

– В результате прерывания от внешнего устройства;

– При наступлении запланированного времени запуска программы.

Операция разблокирования процесса, ожидающего ввода-вывода, происходит следующим образом: когда процессор выполняет некоторый процесс, возникает прерывание от устройства ввода-вывода, сообщающее об окончании операций. Исполняющийся процесс приостанавливается, затем операционная система разблокирует процесс, сделавший запрос на ввод-вывод, и запускает приостановленный или новый процесс, выбранный в ходе планирования.

Переключение контекста

До сих пор мы рассматривали операции над процессами в изоляции друг от друга. Тем не менее, работа мультипрограммной операционной системы представляет собой последовательность операций, выполняемых над различными процессами, что подразумевает переключение процессора с одного процесса на другой.

Для корректного переключения процессора между процессами необходимо сохранить контекст текущего исполняемого процесса и восстано-

вить контекст процесса, на который происходит переключение. Этот процесс сохранения и восстановления работоспособности процессов называется переключением контекста. Время, затраченное на это переключение, представляет собой накладные расходы, которые могут снижать общую производительность системы. Время переключения контекста зависит от вычислительной системы и варьируется от 1 до 1000 микросекунд.

Современные операционные системы уменьшают накладные расходы за счет внедрения расширенной модели процессов, которая включает в себя концепцию нитей исполнения (threads of execution).

В качестве примера мы можем рассмотреть, как на практике происходит операция разблокирования процесса (рис. 5), ожидающего ввода-вывода. Когда процессор выполняет какой-то процесс (предположим, это процесс 1), может возникнуть прерывание от устройства ввода-вывода, сигнализирующее о завершении операций. В этом случае процесс приостанавливается, после чего операционная система разблокирует процесс, который инициировал запрос на ввод-вывод (например, процесс 2), и запускает приостановленный или новый процесс, выбранный в ходе планирования. В итоге, после обработки информации о завершении операции ввода-вывода, выполняется переключение процессов, находящихся в состоянии выполнения.

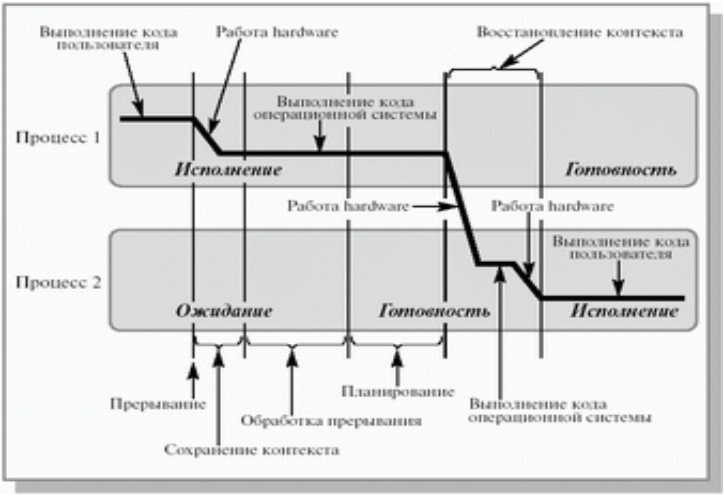


Рис. 5. Выполнение операции разблокирования процесса. Использование термина «код пользователя» не ограничивает общности рисунка только пользовательскими процессами

РАЗДЕЛ 3. ПЛАНИРОВАНИЕ ПРОЦЕССОВ

Тема 3.1. Уровни планирования

Уровни планирования

Говоря о развитии компьютерных систем, можно выделить два вида планирования: планирование заданий и планирование использования процессора.

Планирование заданий

Планирование заданий начало развиваться в пакетных системах, когда для хранения сформированных пакетов заданий стали использоваться магнитные диски. Благодаря концепции прямого доступа, задания можно загружать в компьютере в произвольном порядке, а не только в том, в котором они были записаны на диск. Меняя порядок загрузки заданий, можно повысить эффективность использования системы. Процесс выбора задания для загрузки, а значит, и для создания соответствующего процесса называется планированием заданий.

Планирование использования процессора

Планирование использования процессора впервые было внедрено в мультипрограммных вычислительных системах, где несколько процессов могут одновременно находиться в состоянии готовности. В этом контексте планирование используется для выбора одного процесса, который будет переведен в состояние исполнения.

В вычислительных системах оба вида планирования рассматриваются как уровни планирования процессов.

Планирование заданий как долгосрочное планирование

Планирование заданий считается долгосрочным планированием процессов. Оно отвечает за создание новых процессов в системе, определяя степень мультипрограммирования, то есть сколько процессов может одновременно находиться в системе. Если среднее количество процессов остается постоянным, новые процессы могут появляться только после завершения ранее загруженных. Поэтому долгосрочное планирование происходит довольно редко: между появлением новых процессов могут проходить минуты, а порой даже десятки минут. Решение о запуске того или иного процесса значительно влияет на работу вычислительной системы на длительный срок, что и объясняет название этого уровня планирования.

В некоторых операционных системах долгосрочное планирование сведено к минимуму или отсутствует вовсе. Например, в интерактивных

системах разделения времени процесс может создаваться сразу же после поступления соответствующего запроса. Поддержание разумного уровня мультипрограммирования достигается за счет ограничения числа пользователей, которые могут одновременно работать в системе, а также через психологию пользователей. Если задержка между нажатием клавиши и отображением символа на экране составляет 20–30 секунд, большинство пользователей предпочтет прекратить работу и продолжить её позже, когда система будет менее загружена.

Планирование использования процессора

Планирование использования процессора представляет собой краткосрочное планирование процессов. Оно часто происходит, например, при обращении исполняющегося процесса к устройствам ввода-вывода или по завершении определённого интервала времени. Это краткосрочное планирование осуществляется довольно регулярно, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения влияет на работу системы до следующего аналогичного события, что и обуславливает название этого уровня планирования – краткосрочное.

В некоторых вычислительных системах может быть выгодно временно переместить частично выполненный процесс из оперативной памяти на диск для повышения производительности, а позднее вернуть его обратно для продолжения выполнения. Эта процедура, известная в англоязычной литературе как swapping (или перекачка), решается с помощью промежуточного уровня планирования процессов – среднесрочного.

Планирование процессов включает в себя решение следующих задач:

1. Определить момент времени для смены выполняемого процесса.
2. Выбрать процесс на выполнение из очереди готовых процессов.
3. Переключить контексты «старого» и «нового» процессов.

Первые две задачи решаются программным путем, тогда как последняя в значительной степени осуществляется на аппаратном уровне.

Критерии планирования и требования к алгоритмам

Существует множество различных алгоритмов для каждого уровня планирования процессов. Выбор конкретного алгоритма зависит от класса задач, решаемых вычислительной системой, и целей, которые мы хотим достичь с помощью планирования. Основные **цели** могут включать:

– *Справедливость:* гарантировать каждому заданию или процессу определённую долю времени на использование процессора, избегая ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя не выполняется.

– *Эффективность:* стремиться к тому, чтобы процессор работал на 100% рабочего времени, не позволяя ему простаивать в ожидании готовых

процессов. В реальных системах нагрузка на процессор колеблется от 40% до 90%.

– *Сокращение полного времени выполнения (turnaround time):* минимизировать время от старта процесса или постановки задания в очередь до его завершения.

– *Сокращение времени ожидания (waiting time):* уменьшить время, которое процессы проводят в состоянии готовности и задания в очереди на загрузку.

– *Сокращение времени отклика (response time):* минимизировать время, необходимое процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей, алгоритмы планирования должны также обладать следующими **свойствами:**

– *Предсказуемость:* одно и то же задание должно выполняться примерно за одинаковое время. Применение алгоритма не должно приводить к существенным колебаниям времени исполнения.

– *Минимальные накладные расходы:* алгоритм не должен требовать чрезмерно много времени на определение, какой процесс получит доступ к процессору и на переключение контекстов.

– *Равномерная загрузка ресурсов:* предпочтение следует отдавать процессам, использующим менее загруженные ресурсы.

– *Масштабируемость:* производительность алгоритма не должна значительно ухудшаться при увеличении нагрузки, например, увеличение числа процессов не должно приводить к значительному увеличению полного времени выполнения.

Многие из вышеперечисленных целей и свойств могут конфликтовать друг с другом. Улучшение одного критерия может негативно сказаться на другом, а адаптация алгоритма под один класс задач может дискриминировать задачи другого класса.

Существует множество алгоритмов планирования процессов, каждый из которых по-своему решает вышеупомянутые задачи, преследует различные цели и обеспечивает разное качество мультипрограммирования. Мы рассмотрим две группы наиболее распространенных алгоритмов: алгоритмы, основанные на квантовании, и алгоритмы, основанные на приоритетах.

Алгоритмы на основе квантования

Согласно алгоритмам, основанным на квантовании, смена активного процесса происходит при следующих обстоятельствах:

– Процесс завершился и покинул систему.

– Произошла ошибка.

– Процесс перешел в состояние ожидания.

– Исчерпан квант процессорного времени, отведенный данному процессу.

Процесс и его квантирование

По исчерпании своего квантового времени процесс переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему снова будет предоставлен квант процессорного времени. В это время новый процесс выбирается для исполнения из очереди готовых процессов в соответствии с определёнными правилами. Таким образом, ни один процесс не может занимать процессор длительное время, что делает квантование распространенной практикой в системах разделения времени.

Кванты, выделяемые процессам, могут быть как одинаковыми, так и различными для разных процессов. Они могут иметь фиксированную величину или меняться в разные периоды жизни процесса. Процессы, не использовавшие полностью свой квант (например, из-за перехода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при следующем обслуживании. Очередь готовых процессов может быть организована различными способами: циклически, по принципу «первый пришел – первый обслужился» (FIFO) или «последний пришел – первый обслужился» (LIFO).

Приоритетные алгоритмы

Другая группа алгоритмов опирается на понятие приоритета процесса. Приоритет представляет собой числовое значение, отражающее степень привилегированности процесса в использовании ресурсов вычислительной системы, включая процессорное время: чем выше приоритет, тем больше привилегий имеет процесс.

Приоритет может быть представлен целыми или дробными числами, а также положительными или отрицательными значениями. Чем выше привилегии процесса, тем меньше времени он проводит в очередях. Приоритет может устанавливаться системным администратором в зависимости от важности задачи или рассчитываться самой ОС по определённым правилам. Он может оставаться фиксированным на всю жизнь процесса или изменяться во времени согласно какому-либо закону, в этом случае они называются динамическими.

Существуют две **разновидности** приоритетных алгоритмов: алгоритмы с *относительными* приоритетами и алгоритмы с *абсолютными* приоритетами (рис. 6). В обоих случаях выбор процесса из очереди готовых происходит одинаково: выбирается процесс с наивысшим приоритетом. Однако, в системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ожидания (либо произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса может быть прервано, если в очереди готовых процессов появляется процесс с более

высоким приоритетом. В этом случае прерванный процесс возвращается в состояние готовности.

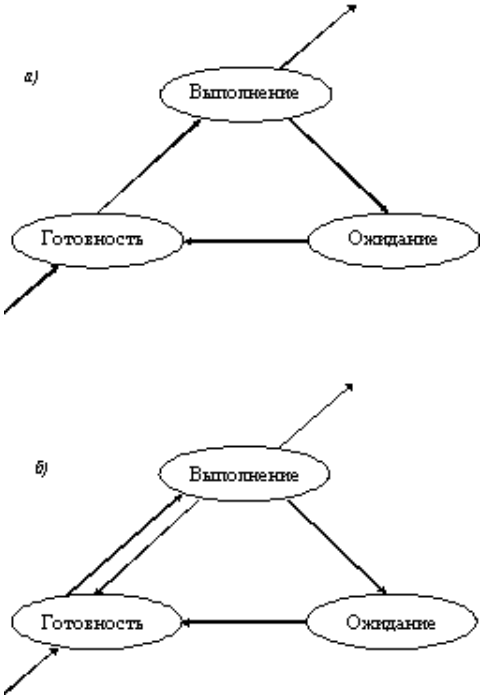


Рис. 6. Графы состояний процессов в системах (а) с относительными приоритетами; (б) с абсолютными приоритетами

Тема 3.2. Вытесняющее и невытесняющее планирование

Планирование и диспетчеризация потоков

В процессе выполнения потоков одного процесса могут возникать множественные прерывания и возобновления. Переход от выполнения одного потока к другому осуществляется через планирование и диспетчеризацию. Процесс определения момента прерывания текущего активного потока и выбора нового потока для выполнения называется планированием. Оно основывается на информации, содержащейся в описателях процессов и потоков.

Планирование потоков включает в себя решение двух основных задач:

1. Определить момент времени для смены текущего активного потока.

2. Выбрать поток для выполнения из очереди готовых потоков.

В большинстве универсальных операционных систем планирование осуществляется динамически (on-line), то есть решения принимаются в процессе работы системы на основе анализа текущей ситуации. В некоторых специализированных системах, таких как системы реального времени, используется статический тип планирования, где набор одновременно выполняемых задач определяется заранее (off-line).

Диспетчеризация означает реализацию решений, найденных в результате планирования, будь то динамическое или статическое. Это включает переключение процессора с одного потока на другой. Прежде чем прервать выполнение потока, операционная система сохраняет его контекст, чтобы впоследствии использовать эту информацию для возобновления работы потока. Контекст отражает состояние аппаратной части компьютера на момент прерывания потока, включая значение счетчика команд, содержимое регистров общего назначения, режим работы процессора, флаги, маски прерываний и другие параметры.

Диспетчеризация и контекст

Во-первых, контекст включает параметры операционной среды, такие как ссылки на открытые файлы, данные о незавершённых операциях ввода-вывода, коды ошибок системных вызовов, выполняемых данным потоком, и т. д. Диспетчеризация состоит из следующих **этапов:**

– Сохранение контекста текущего потока, подлежащего смене.

– Загрузка контекста нового потока, выбранного в результате планирования.

– Запуск нового потока на выполнение.

Вытесняющие и невытесняющие алгоритмы планирования

Вытесняющие алгоритмы

Вытесняющие (preemptive) алгоритмы – это методы, при которых решение о переключении процессора с одного потока на другой принимает операционная система, а не активный поток. В системах с вытесняющим мультипрограммированием вся функциональность планирования потоков сосредоточена в операционной системе. Программист может разрабатывать свои приложения, не беспокоясь о том, что они выполняются одновременно с другими задачами. Операционная система в этом случае:

– Определяет момент, когда активный поток снимается с выполнения.

– Сохраняет его контекст.

– Выбирает следующий поток из очереди готовых и запускает его, загружая соответствующий контекст.

Преимущества: высокая надежность системы в целом.

Недостатки: возможное снижение производительности из-за затрат процессорного времени на переключение задач.

Невытесняющие (non-preemptive) алгоритмы предполагают, что активному потоку разрешается выполняться до тех пор, пока он сам не передаст управление операционной системе для выбора другого готового потока из очереди. В этом механизме планирование распределено между операционной системой и прикладными программами. Приложение, получившее управление, само определяет момент завершения своего выполнения и только после этого передает управление ОС с помощью системного вызова. ОС формирует очереди потоков и выбирает следующий поток для выполнения в соответствии с некоторыми правилами (например, с учетом приоритетов).

Преимущества: высокая производительность и скорость переключения между потоками.

Недостатки: низкая надежность и сложность разработки пользовательских приложений.

Алгоритмы планирования, основанные на приоритетах

Приоритетное обслуживание долговременным потокам предполагает наличие у них заранее определённой характеристики – приоритета, который определяет порядок их выполнения. Приоритет – это числовое значение, характеризующее степень привилегированности потока при использовании ресурсов системы, включая процессорное время: чем выше приоритет, тем больше привилегий у потока и тем меньше времени он будет проводить в очередях.

Приоритет процесса задается операционной системой при его создании. Это значение включается в описатель процесса и используется для назначения приоритета его потокам. При назначении приоритета новосозданному процессу ОС учитывает, системный он или прикладной, статус пользователя, инициировавшего процесс, и возможность явного указания уровня приоритета со стороны пользователя. Поток может быть создан не только по команде пользователя, но и в результате выполнения системного вызова другим потоком.

Изменение приоритета может происходить по инициативе самого потока с помощью обращения к операционной системе или по указанию пользователя, который выполняет соответствующую команду.

Существуют **две разновидности** приоритетного планирования:

1. Обслуживание с относительными приоритетами.
2. Обслуживание с абсолютными приоритетами.

Системы с относительными приоритетами

В системах с относительными приоритетами активный поток выполняется до момента, когда он сам не уступит процессор, перейдя в состояние ожидания, либо произойдёт ошибка или завершение потока. Такой подход минимизирует затраты на переключения между потоками.

Системы с абсолютными приоритетами

В системах с абсолютными приоритетами выполнение активного потока может быть прервано, если в очереди готовых процессов появляется поток с более высоким приоритетом. Это позволяет минимизировать время ожидания потоков в очередях, назначая потокам наивысший приоритет, что дает им возможность вытеснять из процессора все остальные потоки, кроме тех, что имеют такой же высокий приоритет.

Алгоритмы планирования, основанные на квантовании

Многие вытесняющие алгоритмы планирования опираются на концепцию квантования. Согласно этой концепции каждому потоку поочередно предоставляется ограниченный непрерывный период процессорного времени – квант. Смена активного потока происходит в следующих случаях:

– Поток завершил выполнение и покинул систему.

– Произошла ошибка.

– Поток перешел в состояние ожидания.

– Исчерпан квант процессорного времени, выделенный данному потоку.

Когда поток исчерпывает свой квант, он переводится в состояние готовности и ожидает нового квантового времени, в то время как новый поток выбирается для выполнения из очереди готовых.

Кванты, предоставляемые потокам, могут быть одинаковыми для всех потоков или различаться. Если квант небольшой, общее время, которое поток проводит в ожидании процессора, будет прямо пропорционально времени, необходимому для его выполнения (то есть времени исполнения потока при монопольном использовании системы). При увеличении кванта возрастает вероятность завершения потоков в первом цикле выполнения, что делает зависимость времени ожидания потоков от времени их выполнения менее очевидной. При достаточно большом кванте алгоритм квантования становится аналогом алгоритма последовательной обработки, характерного для однопрограммных систем, где время ожидания задачи в очереди не зависит от её длительности.

Кванты, выделяемые одному потоку, могут быть фиксированными или изменяться в разные периоды его жизни. В алгоритмах, основанных на квантовании, не учитывается предварительная информация о задачах. При поступлении задачи на обработку операционная система не имеет данных о её характеристиках, таких как длина или интенсивность запросов к устрой-

ствам ввода-вывода. Дифференциация обслуживания в таких системах основывается на «истории существования» потока.

Алгоритмы планирования

Существует широкий набор алгоритмов планирования, разработанных для достижения различных целей и эффективных для разных классов задач. Многие из них могут применяться на нескольких уровнях планирования.

First-Come, First-Served (FCFS)

Одним из простейших алгоритмов является First-Come, First-Served (FCFS), что переводится как «первым пришел – первым обслужен». Представьте, что процессы в состоянии готовности выстраиваются в очередь. Когда процесс переходит в состояние готовности, его ссылка (PCB) добавляется в конец очереди. Новый процесс выбирается для выполнения из начала этой очереди с удалением ссылки на его PCB. Эта очередь называется FIFO (First In, First Out – «первым вошел – первым вышел»).

Алгоритм FCFS использует невытесняющее планирование: процесс занимает процессор до завершения текущего CPU burst (режима пакетной передачи данных). После этого выбирается следующий процесс для выполнения из начала очереди.

Преимущества FCFS

– Простота реализации.

– Легкость программирования.

– Чёткое соответствие принципу «первым пришел – первым обслужен».

Недостатки FCFS

– Это непланирующий алгоритм; процесс не освобождает CPU, пока не завершит выполнение.

– Высокое среднее время ожидания.

– Короткие процессы, находящиеся в конце очереди, должны ожидать завершения длинных процессов в начале очереди.

– Не идеален для систем с разделением времени.

– Из-за своей простоты FCFS может быть неэффективен.

Пример планирования FCFS

Реальный пример метода FCFS можно увидеть в очереди на кассе кинотеатра. Вот как это работает: человек, прибывший первым, обслуживается первым, и так продолжается, пока последний человек в очереди не купит билет. Этот процесс аналогичен работе процессора в рамках алгоритма FCFS.

Round Robin (RR)

Алгоритм, получивший название Round Robin (RR), представляет собой модификацию алгоритма FCFS. Этот метод также известен как «карусель», поскольку он похож на детскую карусель, где процессы располага-

ются на круге. Каждый процесс получает фиксированный квант времени, обычно от 10 до 100 миллисекунд, в течение которого он может использовать процессор (рис. 7).

Рис. 7. Процессы на карусели

Алгоритм реализуется аналогично FCFS с использованием очереди FIFO для процессов в состоянии готовности. Планировщик выбирает для выполнения процесс, находящийся в начале очереди, и устанавливает таймер, который генерирует прерывание по истечении установленного кванта времени.

Во время выполнения процесса могут возникнуть *два сценария:*

1. Если остаток текущего CPU burst меньше или равен продолжительности кванта времени, процесс завершает своё выполнение раньше времени и освобождает процессор, после чего выбирается новый процесс из начала очереди и таймер перезапускается.

2. Если остаток текущего CPU burst больше, чем квант времени, то по истечении кванта процесс прерывается, помещается в конец очереди, и процессор выделяется следующему процессу из очереди.

Преимущества Round Robin

– Все процессы получают равное распределение ресурсов процессора.

– Процесс выполняется без приоритета.

– Если известно общее количество процессов, можно предположить наихудшее время отклика для каждого процесса.

– После завершения определённого периода выполнения процесс прерывается, и масштаб осуществляется на другой процесс, что способствует более быстрому отклику.

Недостатки Round Robin

– Метод требует больше времени на переключение контекста.

– Эффективность сильно зависит от заданного кванта времени.

– Не позволяет установить приоритеты для процессов.

– Не предоставляет особого преимущества более важным задачам.

– Чем меньше квант времени, тем выше затраты на переключение контекста.

– Определение оптимального кванта времени может быть сложной задачей.

Shortest Job First (SJF)

В процессе анализа алгоритмов FCFS и RR стало очевидно, что порядок процессов в очереди готовых к исполнению имеет большое значение. Если короткие задачи находятся ближе к началу очереди, производительность алгоритмов значительно увеличивается. Если бы была возможность заранее узнать приблизительное время следующих CPU burst для процессов в очереди, можно было бы выбирать для выполнения не первый процесс, а процесс с максимально коротким временем CPU burst. В случае, если несколько процессов имеют одинаковую минимальную длительность, можно использовать FCFS для выбора.

Алгоритм выборки с наименьшим временем выполнения называется Shortest Job First (SJF). Этот алгоритм может быть как вытесняющим, так и невытесняющим.

Вытесняющее и невытесняющее SJF

– *Невытесняющее SJF:* Процессу предоставляется полный доступ к CPU на необходимое время без учета других процессов.

– *Вытесняющее SJF:* В этом варианте учитывается появление новых процессов в очереди готовых к исполнению. Если CPU burst нового процесса короче, чем остаток у действующего, то последний будет прерван.

Основная проблема реализации алгоритма SJF заключается в том, что точное время следующего CPU burst для каждого процесса неизвестно. В пакетных системах пользователи задают предполагаемое время выполнения задач при их создании, что может привести к неэффективному ожиданию. При краткосрочном планировании мы можем лишь прогнозировать длительность следующего CPU burst, основываясь на истории выполнения процесса.

Преимущества SJF

– Долгосрочное планирование: SJF часто используется для долгосрочного планирования.

– Минимальное среднее время ожидания: Метод SJF обеспечивает наименьшее среднее время ожидания при выполнении определенного набора процессов.

– Подходит для пакетных заданий: Алгоритм будет эффективен в случаях, когда время выполнения известно заранее.

– Оценка времени в пакетной системе: В долгосрочном планировании для пакетных систем время выполнения задачи можно получить из описания самой задачи.

– Прогнозирование в краткосрочном планировании: Для краткосрочного планирования необходимо предсказать длительность следующего CPU burst.

Недостатки SJF

– Неопределенность временных характеристик: Необходимость заранее знать время завершения процессов затрудняет реализацию, так как это трудно спрогнозировать.

– Долгосрочное планирование ограниченно: SJF часто применяется в пакетных системах для долгосрочного планирования.

– Проблемы в краткосрочном планировании: Алгоритм не подходит для краткосрочного планирования, так как нет надежных методов для предсказания длины предстоящего CPU burst.

– Долгие задержки: Этот алгоритм может приводить к значительным задержкам выполнения задач или недостаточному времени их завершения.

Гарантированное планирование

Для интерактивной работы N пользователей в вычислительной системе может использоваться алгоритм планирования, который гарантирует каждому пользователю ~1/N части процессорного времени. Однако такой подход имеет свои недостатки: предугадать поведение пользователей сложно. Если, к примеру, один из пользователей уйдет на длительный перерыв, его процессы могут занять неоправданно много процессорного времени при его возвращении.

Приоритетное планирование

Алгоритмы SJF и гарантированного планирования могут рассматриваться как частные случаи приоритетного планирования. В этом случае каждому процессу присваивается числовое значение – приоритет, на основе которого выделяется процессор. Процессы с одинаковыми приоритетами обрабатываются по принципу FCFS. В алгоритме SJF в качестве приоритета используется оценка продолжительности следующего CPU burst: меньшая длительность означает более высокий приоритет. В гарантированном планировании приоритет определяется вычисленным коэффициентом справедливости: чем он меньше, тем выше приоритет у процесса.

Алгоритмы приоритетного планирования могут учитывать как внутренние параметры (например, ограничения по времени использования процессора, объем памяти, количество открытых файлов и устройств ввода-вывода, соотношение средних длительностей I/O burst к CPU burst), так

и внешние факторы (например, важность процесса для достижения целей, стоимость процессорного времени и политические факторы). SJF и гарантированное планирование используют внутренние параметры.

Планирование с приоритетами может быть как вытесняющим, так и невытесняющим: при вытеснении процесс с более высоким приоритетом может прервать исполняющийся процесс с более низким приоритетом, тогда как в невытесняющем варианте он просто добавляется в начало очереди готовых процессов.

Статические и динамические приоритеты

Статические приоритеты не меняются с течением времени и легко реализуются, однако они не учитывают изменений в состоянии системы. Динамические приоритеты адаптируются во время выполнения процессов и могут измениться в зависимости от различных событий (например, при создании нового процесса, разблокировке, завершении работы или по истечении определенного кванта времени).

Примеры алгоритмов с динамическими приоритетами включают SJF и гарантированное планирование. Хотя схемы с динамической приоритетностью сложнее в реализации и требуют больше ресурсов, их использование может значительно повысить эффективность работы системы.

Проблемы приоритетного планирования

Главная проблема приоритетного планирования заключается в риске, что низкоприоритетные процессы могут оставаться без внимания на неопределенно долгий срок, если механизм назначения и изменения приоритетов выбран неправильно. Обычно возникают два сценария: либо эти процессы все же дождутся своей очереди на выполнение, либо система может быть вынуждена выключиться, что приведет к потере этих процессов. Эта проблему можно решить, увеличивая значения приоритета процессов, находящихся в состоянии готовности, с течением времени. Например, начальные приоритеты процессов могут варьироваться от 128 до 255, и по истечении определенного времени их значения приоритетов будут уменьшаться на 1. Процесс, который находился в состоянии выполнения, получает свое первоначальное значение приоритета. Даже такая простая схема обеспечивает справедливое распределение времени выполнения для всех процессов.

Алгоритм многоуровневых очередей (Multilevel Queue)

Для систем, где процессы классифицируются в разные группы, были разработаны алгоритмы на основе многоуровневых очередей. Для каждой группы процессов создается своя очередь в состоянии готовности с фиксированными приоритетами. Например, очередь системных процессов может иметь более высокий приоритет, чем очередь пользовательских процессов, что гарантирует, что ни один пользовательский процесс не будет выбран для выполнения, пока есть готовый системный процесс. Внутри очередей

можно применять различные алгоритмы планирования: для более длинных процессов, не требующих взаимодействия с пользователем, можно использовать FCFS, а для интерактивных – RR. Такой подход повышает гибкость планирования, позволяя применять наиболее подходящий алгоритм для процессов с различными характеристиками.

Алгоритм многоуровневых очередей с обратной связью (Multilevel Feedback Queue)

Следующим шагом в развитии многоуровневых очередей стало добавление механизма обратной связи, который позволяет процессам мигрировать между очередями в зависимости от их поведения. Планирование осуществляется на основе вытесняющего приоритетного механизма: процессы в более высокоприоритетной очереди могут вытеснять процессы из очередей с более низким приоритетом. Процессы в очереди 1 не могут выполняться, пока есть хотя бы один процесс в очереди 0, рассматривая процесс в очереди 2, если в очередях 0 и 1 также есть процессы. Если в ходе выполнения процесса появляется новый процесс в более приоритетной очереди, текущий процесс прерывается.

При поступлении в исполнение новые процессы попадают в очередь 0 и получают квант времени, к примеру, в 8 единиц. Если их CPU burst меньше этого кванта, они остаются в очереди 0. В противном случае они переходят в очередь 1, где квант времени составляет 16 единиц. Если процесс не укладывается в это время, он перемещается в очередь 2 с квантованием времени в 32 единицы. Если и этого времени не хватает, процесс попадает в очередь 3, где кванты времени не регулируются, и он может исполняться до завершения своего CPU burst при отсутствии готовых процессов в других очередях. Таким образом, анализируется время работы процессов: краткие задачи имеют высокий приоритет и размещаются в верхние очереди, в то время как более длительные процессы с низкими запросами к времени отклика попадают в низкоприоритетные очереди.

Возможна миграция процессов в обратном направлении. Например, после завершения операций ввода с клавиатуры процессы из очередей 1, 2 и 3 могут снова попадать в очередь 0. После завершения операций ввода-вывода процессы из очереди 3 могут перемещаться в очередь 1, а после завершения ожидания событий – из очереди 3 в очередь 2. Возможность перемещения процессов из низкоприоритетных очередей в высокие позволяет учитывать изменения поведения процессов с течением времени.

Многоуровневые очереди с обратной связью представляют собой наиболее универсальный подход к планированию процессов. Хотя их реализация сложна, они обеспечивают максимальную гибкость. Для конкретного внедрения этого подхода *необходимо определить:*

– количество очередей процессов в состоянии готовности;

– алгоритм планирования между очередями;
– алгоритмы планирования внутри очередей;
– правила назначения новых процессов в очереди;
– правила перевода процессов между очередями.

Изменение любого из этих аспектов может существенно повлиять на поведение вычислительной системы.

РАЗДЕЛ 4. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ И ОСНОВНЫЕ АСПЕКТЫ ЕГО ЛОГИЧЕСКОЙ ОРГАНИЗАЦИИ

Тема 4.1. Взаимодействующие процессы

Во всех вычислительных системах процессы регулярно ожидают своей очереди на доступ к процессору и ведут конкуренцию за доступные ресурсы. Операционная система, в свою очередь, стремится изолировать каждый процесс, обеспечивая его собственным адресным пространством, а также, при возможности, выделяя отдельные ресурсы. Попытка одного процесса выйти за границы своего адресного пространства, как правило, приводит к аварийному завершению его работы. Однако в ряде случаев процессы могут взаимодействовать для выполнения общей задачи.

Взаимодействие процессов

Для достижения определённых целей, процессы, даже запущенные разными пользователями, могут исполняться параллельно на разных машинах либо псевдопараллельно – на одной, при этом координируя свои действия между собой. Это взаимодействие необходимо в следующих *случаях:*

• *Для повышения эффективности работы.* Пока один из процессов находится в режиме ожидания какого-либо события, например завершения операции ввода-вывода, другие процессы могут выполнять полезные вычисления, тем самым ускоряя общее выполнение задачи. В системах с несколькими процессорами программы могут быть разбиты на части, каждая из которых запускается на своём процессоре, обеспечивая реальное параллельное выполнение.

• *Для совместного доступа к данным.* Разные процессы могут обрабатывать одну и ту же базу данных или работать с общим файлом, изменяя его содержимое синхронно и согласованно.

• *При модульной архитектуре систем.* Например, в операционных системах, построенных по микроядерной архитектуре, различные модули ОС реализуются в виде отдельных процессов, которые взаимодействуют путём обмена сообщениями через микроядро.

• *Для повышения удобства использования.* Например, пользователь может одновременно редактировать и отлаживать программу. Для этого процесс редактора должен быть способен взаимодействовать с процессом отладчика.

Обмен информацией между процессами – ключевой элемент их взаимодействия. Если процесс не изменяет своё поведение в ответ на полученные данные, значит, в реальности никакого взаимодействия не происходит. Только те процессы, чьи действия зависят от информации, полученной от других, считаются взаимодействующими, или кооперативными. Процессы, которые полностью игнорируют существование других, называются независимыми.

С точки зрения операционной системы, процессы изначально изолированы. Они не должны нарушать работу друг друга. Разделение ресурсов и адресного пространства обеспечивается именно с этой целью. Поэтому для организации корректного обмена данными необходимы специальные механизмы, реализуемые на уровне ОС. Простая передача значения из одного процесса в область памяти другого невозможна без соответствующих действий, поскольку это нарушает правила безопасности и стабильности системы. Далее будут рассмотрены базовые принципы организации совместной работы процессов.

Категории средств обмена информацией между процессами

Взаимодействие процессов всегда реализуется через обмен информацией. Все существующие методы обмена данными можно условно классифицировать по объему передаваемой информации и степени воздействия на поведение процессов на *три группы:*

• *Сигнальные средства.* Передают минимальный объём информации – как правило, один бит, сигнализирующий о наступлении события («да» или «нет»). Эти средства используются, в основном, для уведомления процесса о каком-либо происшествии. Влияние сигнала на поведение процесса зависит от того, как он обрабатывает эту информацию. Если сигнал проигнорирован или интерпретирован неверно, это может привести к сбоям в работе всей системы.

• *Канальные средства.* Эти механизмы предполагают наличие логических каналов связи между процессами, через которые они обмениваются сообщениями. Такой обмен можно сравнить с телефонным разговором, письменной перепиской или публикацией объявлений. Количество передаваемой информации ограничено пропускной способностью канала. Чем больше данных передаётся, тем больше влияние процесса-отправителя на поведение процесса-получателя.

• *Разделяемая память.* Некоторые процессы могут использовать одну и ту же область оперативной памяти. Обычно такие области создаются по запросу с участием операционной системы. Это можно сравнить с проживанием нескольких студентов в одной комнате – они имеют общий доступ ко всему, что в ней находится, но любое неосторожное действие может создать конфликт. Данный способ обеспечивает максимальную скорость об-

мена данными и влияние на поведение другого процесса, но требует особой осторожности. Работа с разделяемой памятью осуществляется средствами языков программирования. В отличие от сигнальных и канальных методов, где используются системные вызовы, здесь используется прямой доступ к памяти.

Логическая организация взаимодействия

При анализе способов коммуникации между процессами акцент делается не на физических аспектах (как, например, использование общей шины или прерываний), а на логической структуре взаимодействия. Именно логическая организация определяет, как именно используются те или иные средства связи. Некоторые аспекты логики взаимодействия характерны для всех методов, а некоторые – уникальны для отдельных типов обмена.

Потоки исполнения

Обсуждаемые выше механизмы относятся к взаимодействию самостоятельных процессов. Однако в рамках развития операционных систем появились новые подходы, в частности – концепция потоков исполнения, которые изменили само понимание процесса.

Внедрение мультипрограммирования позволило повысить производительность систем, сокращая общее время ожидания выполнения задач. Но при этом отдельный процесс не мог быть завершён быстрее, чем в однопрограммной среде на том же оборудовании. Если же задача обладает внутренним параллелизмом, её выполнение можно ускорить путём одновременного исполнения нескольких частей с помощью взаимодействующих компонентов.

Так возникло понятие **потока исполнения** (англ. thread) – логической единицы выполнения внутри процесса. Все потоки процесса имеют общий код, глобальные переменные и доступ к системным ресурсам, но каждый поток обладает своим собственным счётчиком команд, регистровым набором и стеком. Таким образом, процесс становится совокупностью потоков и ресурсов, которыми они пользуются.

Если процесс содержит только один поток – он эквивалентен традиционному (однопоточному) процессу. Потоки нередко называют «лёгкими процессами», поскольку по своим свойствам они близки к процессам, но создаются и переключаются гораздо быстрее.

Каждый поток может порождать дочерние потоки – но только внутри одного и того же процесса. Потоки имеют такие же состояния, как и процессы: ожидание, готовность, исполнение, завершение. Процесс считается:

• *в состоянии готовности,* если хотя бы один поток готов к исполнению, но ни один не выполняется;

• *в состоянии выполнения,* если хотя бы один поток выполняется;

• *в состоянии ожидания,* если все потоки приостановлены;

• *в состоянии завершения,* если все потоки завершили выполнение.

Когда один поток блокируется, другие потоки того же процесса могут продолжать выполнение. Потоки конкурируют за процессорное время так же, как это делают независимые процессы.

Так как потоки используют общее адресное пространство и ресурсы, их создание и переключение между ними обходится значительно дешевле по времени и ресурсам, чем при работе с полноценными процессами.

Операционные системы реализуют поддержку потоков на *двух уровнях:*

• *На уровне ядра.* Ядро ОС оперирует потоками напрямую, производит планирование их выполнения, а управление ресурсами осуществляется на уровне процессов. Это полноценная реализация многопоточности.

• *На уровне пользовательских библиотек.* Ядро не имеет информации о потоках, планирование и управление ими осуществляется внутри процесса средствами библиотек. В этом случае блокировка одного потока приводит к блокировке всего процесса, так как ОС воспринимает его как единое целое. Подобная модель лишь эмулирует многопоточность и не позволяет эффективно использовать все преимущества.

РАЗДЕЛ 5. СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

Тема 5.1. Проблемы синхронизации

Сущность проблемы синхронизации

При выполнении задач в операционной системе нередко требуется, чтобы процессы взаимодействовали друг с другом. Например, один процесс может пересылать данные другому, или несколько процессов могут одновременно использовать общий файл. Во всех таких ситуациях возникает необходимость синхронизировать выполнение процессов, чтобы избежать конфликтов и обеспечить корректность работы. Решение этой задачи может включать в себя приостановку и возобновление выполнения процессов, формирование очередей, а также блокировку и освобождение общих ресурсов.

Понятие критической секции

Центральным элементом синхронизации является так называемая критическая секция – это участок кода, где происходит доступ к разделяемым данным или ресурсам. Чтобы избежать проблем, таких как состояние гонки, необходимо гарантировать, что одновременно критическую секцию, связанную с определённым ресурсом, может выполнять только один процесс. Такая организация называется взаимным исключением.

Наиболее примитивный способ достижения взаимного исключения – это запрет прерываний для процесса, находящегося в критической секции. Однако это решение нельзя считать надёжным, поскольку пользовательский процесс получает полный контроль над системой, что может привести к её «зависанию», если произойдёт сбой – прерывания не будут разрешены, и система перестанет отвечать.

Другой, более безопасный подход – это применение блокирующих переменных. Каждому общему ресурсу сопоставляется двоичная переменная: если её значение равно 1 – ресурс свободен, если 0 – занят. Перед входом в критическую секцию процесс проверяет значение переменной: если ресурс недоступен, он повторяет проверку до тех пор, пока тот не освободится. Как только переменная указывает на доступность, процесс устанавливает её в 0 и входит в критическую секцию. После завершения работы с ресурсом он возвращает значение переменной в 1, обозначая, что ресурс снова доступен.

Важно, чтобы операция проверки и изменения значения блокирующей переменной была атомарной, то есть неделимой. Идеально, если архитектура процессора предоставляет специальную инструкцию для такой операции, либо система должна обеспечивать неделимость с помощью программных примитивов, которые временно отключают прерывания на время выполнения.

Несмотря на то, что метод блокирующих переменных обеспечивает взаимное исключение, у него есть серьёзный недостаток: пока один процесс работает в критической секции, другой тратит ресурсы процессора на постоянную проверку состояния переменной, не выполняя полезной работы. Это ведёт к неэффективному использованию процессорного времени.

Состояния взаимных блокировок (тупики)

Рассмотренный пример помогает перейти к следующей важной проблеме синхронизации – взаимной блокировке процессов. Это явление также называют тупиком (deadlock), клинчем или взаимной блокировкой.

В зависимости от того, с какой скоростью работают различные процессы, они могут либо бесконфликтно использовать совместные ресурсы, либо выстраиваться в очередь к ним, либо входить в состояние, при котором каждый ожидает освобождения ресурса, занятого другим. В последнем случае наступает тупик. Он отличается от обычной очереди тем, что в очереди ресурсы в итоге становятся доступными, и процесс продолжает выполнение, а в тупике ресурсы остаются заблокированными бесконечно.

Проблематика тупиков включает в себя *три аспекта:*

1. Предотвращение тупиков;
2. Обнаружение тупиков;
3. Восстановление после возникновения тупиковых состояний.

Предупреждение тупиков возможно ещё на стадии разработки программного обеспечения. Программы следует проектировать так, чтобы вне зависимости от времени выполнения и скорости процессов тупиковая ситуация не могла возникнуть. Например, если все процессы запрашивают ресурсы в строго определённой последовательности, вероятность тупика устраняется полностью.

Альтернативный метод предотвращения тупиков – динамический. Он реализуется в виде специальных правил распределения ресурсов. Одно из таких правил: ресурсы назначаются в фиксированном порядке, одинаковом для всех процессов.

Существуют также формализованные методы обнаружения тупиков, основанные на построении таблиц, отображающих текущее распределение ресурсов и активные запросы. Анализ таких таблиц позволяет определить, существуют ли циклы зависимостей, свидетельствующие о взаимной блокировке.

Если тупик всё же возник, полное завершение всех заблокированных процессов не всегда необходимо. Возможны менее радикальные меры, например: завершение части процессов для освобождения ресурсов, возврат некоторых задач в область подкачки (swap), либо «откат» процессов до заранее определённых контрольных точек, где сохранена информация, необходимая для безопасного продолжения выполнения после устранения тупика. Эти контрольные точки должны быть расставлены в коде в тех местах, после которых существует риск возникновения взаимной блокировки.

РАЗДЕЛ 6. ОРГАНИЗАЦИЯ ПАМЯТИ КОМПЬЮТЕРА. ПРОСТЕЙШИЕ СХЕМЫ УПРАВЛЕНИЯ ПАМЯТЬЮ

Тема 6.1. Типы адресов. Методы распределения памяти без привлечения дискового пространства

Операционная система (ОС) выполняет ряд ключевых функций в процессе управления оперативной памятью. Среди них – контроль за занятыми и свободными участками памяти, предоставление памяти активным процессам, ее освобождение по завершении работы этих процессов, временное перемещение некоторых процессов на диск при нехватке оперативной памяти, а также возврат этих процессов в память при появлении свободного места. Кроме того, ОС отвечает за сопоставление адресов в программе с конкретными физическими адресами в оперативной памяти.

Типы адресов

Для обозначения данных и команд применяются три типа адресов: символьные имена (метки), виртуальные адреса и физические адреса.

• *Символьные имена* создаются программистом при написании кода на языке программирования или ассемблере.

• *Виртуальные адреса* назначаются транслятором при преобразовании программы в машинный код. Поскольку в момент трансляции обычно неизвестно, куда именно в оперативной памяти будет загружена программа, виртуальные адреса присваиваются условно, как правило, начиная с адреса 0. Набор таких адресов образует *виртуальное адресное пространство* процесса. Каждый процесс получает собственное виртуальное адресное пространство, максимальный размер которого определяется разрядностью адреса в конкретной архитектуре. Как правило, это пространство значительно превышает объем физической памяти.

• *Физические адреса* – это реальные номера ячеек в оперативной памяти, где фактически размещаются команды и данные.

Существует **два подхода** к преобразованию виртуальных адресов в физические:

1. *С использованием перемещающего загрузчика.* Эта специальная системная программа выполняет трансляцию виртуальных адресов в физические при загрузке, учитывая начальный физический адрес размещения и сведения о адресно-зависимых элементах кода. Преобразование происходит однократно – при загрузке.

2. *С динамической трансляцией во время выполнения.* Программа загружается в память без изменений, и ОС лишь фиксирует смещение между виртуальными и физическими адресами. Преобразование виртуального адреса в физический выполняется при каждом обращении к памяти. Этот способ более гибкий, поскольку позволяет перемещать программу по памяти во время ее работы, однако требует дополнительных вычислительных ресурсов из-за постоянных преобразований.

В отдельных случаях, например в специализированных системах с заранее известным адресом загрузки, транслятор может сразу сформировать код с физическими адресами.

Методы распределения памяти без использования диска

Существуют *два основных класса* методов управления памятью:

• *с перемещением* процессов между ОЗУ и диском;

• *без такого перемещения.* Рассмотрим второй, более простой вариант.

1. Фиксированные разделы

Наиболее простой подход – *разделение памяти на области фиксированного размера.* Это деление может быть произведено вручную при запуске системы или заранее в момент ее настройки. При поступлении задачи она либо помещается в общую очередь, либо направляется в очередь определенного раздела.

Система управления памятью выполняет следующие *действия:*

• сравнивает размер задачи с размерами доступных разделов;

• выбирает подходящий раздел;

• производит загрузку и настройку адресов.

Основным достоинством этого метода является его простота. Однако он имеет важное ограничение: *жесткая структура.* В каждом разделе может работать только одна задача, независимо от её реального объема. Это ограничивает количество одновременно выполняемых процессов. Кроме того, даже если программа небольшая, она занимает весь раздел, что снижает эффективность использования памяти. Также возможна ситуация, когда программа не запускается из-за неподходящего размера разделов, несмотря на наличие достаточной общей памяти.

2. Переменные разделы

В этом методе *вся память изначально свободна,* и разделы создаются *по мере поступления задач.* Каждой задаче выделяется ровно столько памяти, сколько ей необходимо. Если требуемого объема нет, задача ожидает в очереди. После завершения задачи освобожденный участок памяти может быть использован под другую задачу.

Операционная система в этом случае:

• ведет таблицы свободных и занятых участков, включая их начальные адреса и размеры;

• при поступлении новой задачи ищет в таблице подходящий свободный участок;

• загружает задачу в выделенный участок и обновляет таблицы;

• после завершения задачи – снова обновляет информацию о свободной и занятой памяти.

Адреса в программах настраиваются один раз – при загрузке, через перемещающий загрузчик, так как программы *не перемещаются* в процессе выполнения.

Существуют различные *стратегии выбора* подходящего свободного участка:

• первый подходящий по размеру;

• минимально достаточный;

• максимально возможный среди подходящих.

Каждая стратегия имеет свои плюсы и минусы.

Главный недостаток метода – *фрагментация памяти,* при которой в системе остается множество небольших несмежных свободных участков, в которые нельзя загрузить ни одну из задач. При этом общий объем фрагментов может быть значительным.

3. Перемещаемые разделы (метод сжатия)

Для устранения фрагментации применяется *перемещение занятых участков памяти* в одну сторону (к младшим или старшим адресам), чтобы освободить один непрерывный блок памяти. ОС, помимо обычных функций, должна периодически копировать содержимое разделов и пересчитывать таблицы. Этот процесс называется *сжатием памяти.*

Сжатие может проводиться:

• после завершения каждой задачи;

• только в случае, если отсутствует достаточный непрерывный участок для новой задачи.

Первый вариант требует меньше вычислений при пересчете таблиц, а второй – реже выполняет саму процедуру сжатия. Однако поскольку код программы перемещается во время выполнения, преобразование адресов должно быть *динамическим.*

Несмотря на эффективность использования памяти, процедура сжатия может оказаться ресурсоемкой по времени, что нередко сводит на нет её преимущества.

Подсистема управления памятью в операционной системе

Память в вычислительной системе – это один из ключевых ресурсов, требующий точного и эффективного распределения, особенно в условиях работы мультипрограммной операционной системы. Её важность обусловлена тем, что центральный процессор способен обрабатывать инструкции только при условии, что они размещены непосредственно в оперативной

памяти. Ресурс памяти распределяется между прикладными программами, а также внутренними компонентами самой операционной системы.

Операционная система выполняет следующие функции в рамках управления памятью в многозадачной среде:

• ведёт контроль над занятой и свободной памятью;

• выделяет память процессам при их запуске и освобождает её после завершения выполнения;

• осуществляет перенос (вытеснение) данных и команд процессов с оперативной памяти на диск при нехватке физической памяти, а затем возвращает их обратно, когда появляется свободное место;

• выполняет адаптацию (настройку) адресов программы в соответствии с выделенной областью физической памяти.

Помимо первоначального выделения области памяти при создании процесса, операционная система также занимается динамическим управлением – она должна удовлетворять запросы приложений на получение дополнительной памяти в процессе их выполнения. Когда приложению больше не требуется выделенная память, оно освобождает её, возвращая обратно операционной системе. Такие произвольные запросы на распределение памяти различной длины и в различные моменты времени приводят к фрагментации – образованию разрозненных неиспользуемых участков, что снижает общую эффективность использования памяти. В этом случае ОС также выполняет функцию дефрагментации.

Кроме того, важной задачей системы управления памятью является защита. Операционная система должна предотвращать попытки одного процесса обращаться к данным другого. Механизмы защиты памяти реализуются программными средствами ОС при участии аппаратных компонентов компьютера, что обеспечивает безопасность и стабильность выполнения программ.

Тема 6.2. Распределение памяти с использованием дискового пространства

Понятие виртуальной памяти

Во время жизненного цикла программы для определения переменных и команд используются разные типы обозначений: символьные имена (или метки), виртуальные и физические адреса (рис. 8).

Символьные имена задаются программистом при написании кода на языках программирования высокого уровня или на языке ассемблера.

Виртуальные адреса, которые также называют логическими или математическими, формируются транслятором при преобразовании программы в машинный код.

Физические адреса – это конкретные номера ячеек оперативной памяти, в которых в действительности размещаются команды и данные.

Вся совокупность логических адресов, используемых отдельным процессом, образует его **виртуальное адресное пространство.** Этот диапазон адресов одинаков для всех процессов, однако каждый процесс имеет индивидуальное виртуальное пространство. Транслятор назначает виртуальные адреса каждой программе отдельно и независимо от других.

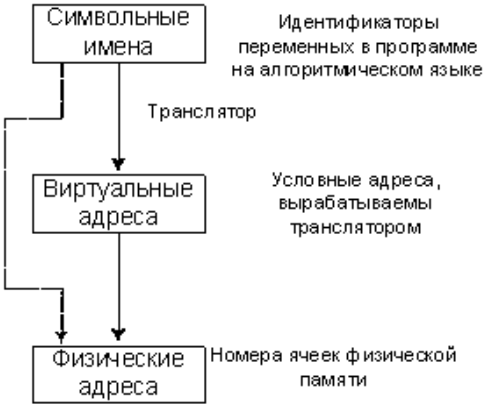


Рис. 8. Типы обозначений во время жизненного цикла программы

Виртуальная память – это технология, при которой части исполняемых процессов (программы, находящиеся в работе) могут временно перемещаться из оперативной памяти на диск и обратно. Таким образом, между диском и ОЗУ происходит постоянное перемещение фрагментов процессов, таких как страницы или сегменты.

Размер виртуального адресного пространства ограничивается только архитектурной разрядностью системы и обычно превышает объем физической оперативной памяти.

Методы распределения памяти

Для выполнения программы необходимо, чтобы она находилась в оперативной памяти, так как только оттуда процессор способен извлекать ко-

манды и обрабатывать их. Все методы управления памятью делятся на две основные группы:

1. Алгоритмы, при которых используется перенос сегментов между ОЗУ и диском.

2. Алгоритмы, не использующие внешнюю память (рис. 9).

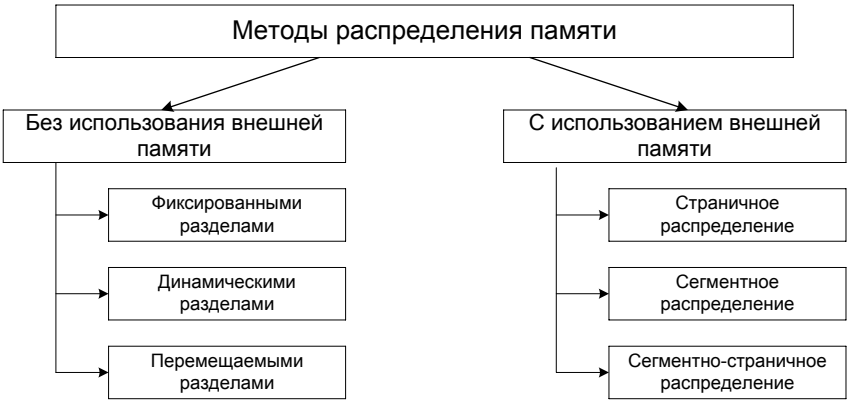


Рис. 9. Классификация методов распределения памяти

Распределение памяти фиксированными разделами

Этот подход предполагает разделение оперативной памяти на несколько фиксированных участков – *разделов,* размеры которых задаются заранее, обычно вручную, при запуске или установке системы. После задания границ разделов они не изменяются.

Каждый новый процесс при запуске либо становится в общую очередь, либо попадает в очередь конкретного раздела. Управляющая подсистема памяти выполняет следующие функции:

• Сравнивает запрашиваемый объем памяти с имеющимися свободными разделами и выбирает подходящий.

• Загружает программу в выбранный раздел, настраивая адреса в коде.

Иногда разработчик может заранее указать раздел, в котором программа должна быть выполнена, что позволяет сразу скомпилировать код под конкретную область памяти.

Преимущество – простота реализации.

Недостаток – ограниченная гибкость: каждый раздел обслуживает только один процесс, а общее число процессов ограничено числом разделов.

Такой метод применим в **системах реального времени,** где важно минимизировать накладные расходы.

Распределение памяти динамическими разделами

В этом методе вся оперативная память для приложений изначально свободна. При запуске процессу выделяется вся требуемая память (при условии её наличия). Если памяти недостаточно – выполнение процесса невозможно. По завершении выполнения память освобождается, и она становится доступной для других программ.

Операционная система при этом:

1. Ведёт таблицы свободных и занятых участков с указанием начальных адресов и размеров.

2. При запуске нового процесса анализирует таблицы, выбирая подходящий участок памяти. Выбор может осуществляться по алгоритмам:

– Первый подходящий;

– С наименьшим подходящим размером;

– С наибольшим подходящим размером.

3. Загружает код в выделенное пространство, обновляя таблицы.

4. При завершении процесса – освобождает память и корректирует таблицы.

Плюс – высокая гибкость по сравнению с фиксированными разделами.

Минус – фрагментация памяти: множество мелких свободных участков, которые в сумме дают большой объем, но не подходят для размещения новых процессов.

Распределение памяти перемещаемыми разделами

Ключевая особенность – применение дефрагментации, при которой занятые участки перемещаются в сторону начала или конца памяти. Это позволяет объединить разрозненные фрагменты в одну непрерывную область.

Операционная система в этом случае:

1. Ведёт таблицы занятых и свободных блоков памяти.

2. При запуске процесса анализирует потребность в памяти и ищет подходящий участок.

3. Загружает код в выбранное место, обновляя таблицы.

4. После завершения процесса – освобождает память и актуализирует таблицы.

Дополнительно система выполняет сжатие (перемещение фрагментов памяти) либо после каждого завершения процесса, либо только при нехватке памяти для нового процесса.

Плюсы – более эффективное использование памяти.

Минусы – снижение производительности, так как сжатие требует значительных ресурсов.

Страничное распределение памяти

Этот метод основан на делении виртуального адресного пространства процесса на *страницы* фиксированного размера (обычно степень двойки

– 512, 1024, 4096 байт и т. д.). Оперативная память также делится на *физические страницы* (или кадры) того же размера.

При запуске процесса в оперативную память загружаются начальные страницы. Все виртуальные страницы находятся на диске, и могут быть перемещены в оперативную память по мере необходимости. Виртуальные страницы процесса могут размещаться в оперативной памяти несмежно.

Для каждого процесса создается *таблица страниц,* где для каждой виртуальной страницы указывается:

• Номер физической страницы;

• Признак присутствия (в памяти или нет);

• Признак модификации (была ли страница изменена);

• Признак обращения (бит доступа).

При обращении к памяти:

• Осуществляется поиск записи о виртуальной странице;

• При наличии страницы в памяти выполняется трансляция адреса;

• Если страницы нет – возникает *страничное прерывание:* процесс приостанавливается, активируется другой, а нужная страница загружается с диска;

• Если свободных страниц в ОЗУ нет, ОС выбирает, какую страницу выгрузить. При этом, если страница была изменена – она сохраняется на диск.

Преимущество – возможность эффективно работать с большими объемами памяти и избегать фрагментации.

Недостаток – более сложная реализация.

Сегментное распределение памяти

В этом методе виртуальное адресное пространство делится на сегменты, каждый из которых логически связан с определенным компонентом программы – например, отдельной функцией или массивом данных.

Максимальный размер сегмента определяется разрядностью адреса (например, при 32-битной архитектуре – до 4 ГБ).

При запуске процесса в ОЗУ загружается только часть сегментов, тогда как остальная информация хранится на диске. Смежные сегменты виртуальной памяти могут размещаться в ОЗУ на несмежных участках.

При обращении к отсутствующему в памяти сегменту возникает прерывание, процесс приостанавливается, и система загружает необходимый сегмент с диска. Если нет места в памяти, один из сегментов выгружается.

Плюс – логичное и понятное представление программы в памяти, удобное для компилятора и программиста.

Минус – необходимость поиска подходящего по размеру непрерывного блока памяти для загрузки сегмента.

Тема 6.3. Иерархия запоминающих устройств. Принцип кэширования дисков

Иерархическая структура запоминающих устройств

Память вычислительных систем представляет собой иерархически организованный комплекс различных запоминающих устройств. В эту иерархию входят: регистры процессора, разнообразные типы сверхоперативной и оперативной памяти, а также внешние носители – диски, магнитные ленты и другие. Эти устройства различаются между собой как по времени доступа к данным, так и по стоимости хранения информации в пересчёте на один бит (рис. 10).

С точки зрения пользователя, идеальной была бы память, сочетающая низкую стоимость с высокой скоростью доступа. Однако такое сочетание трудно достижимо. Кэш-память представляет собой компромиссное решение, позволяющее уравновесить эти два критерия.

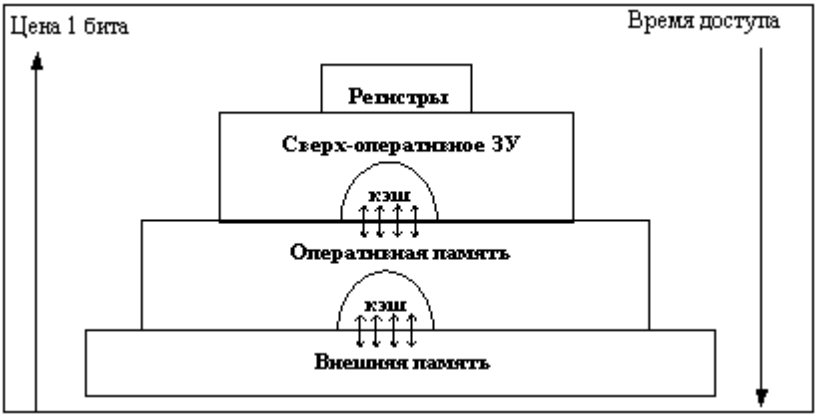


Рис. 10. Иерархия ЗУ

Уровни памяти в современной вычислительной системе

В развитых вычислительных системах можно выделить *несколько уровней памяти:*

1. *Регистровая память* – встроенные в процессор запоминающие устройства, к которым осуществляется максимально быстрый доступ. Использование регистров позволяет сократить количество обращений к более медленным уровням памяти, расположенным вне процессора.

2. *Кэш-память, или сверхоперативное ЗУ* – это быстродействующая память, предназначенная для временного хранения копий данных из основной памяти. Она может размещаться как внутри процессора, так и вне его

корпуса. Основное назначение – ускорение доступа к наиболее часто используемой информации.

3. *Оперативная память (ОЗУ, RAM)* – основное запоминающее устройство, служащее для хранения программ, выполняемых пользователем, а также промежуточных данных и результатов вычислений.

4. *Постоянная память (ПЗУ, ROM)* – элемент основной памяти, предназначенный для хранения данных, редко подверженных изменениям. Примеры таких данных: начальные коды загрузки, тестовые программы и микропрограммы управления.

5. *Специализированные виды памяти,* например, видеопамять, используются для хранения данных, отображаемых на экране монитора или используемых другими периферийными устройствами.

6. *Внешняя память* – включает в себя магнитные и оптические диски, а также флеш-накопители. Она применяется для долговременного хранения большого объема информации, выходящего за рамки возможностей оперативной памяти.

Механизм кэширования дисков

Кэширование – это метод организации совместной работы двух уровней памяти, отличающихся по скорости доступа и стоимости хранения информации. Основная цель кэширования – сократить среднее время доступа к данным за счет временного размещения часто используемой информации из медленного хранилища в более быстрое.

Термин «кэш-память» может относиться как к самому методу, так и к быстрому запоминающему устройству, в котором хранятся дублирующие копии данных. Такое устройство имеет сравнительно небольшой объём и более высокую стоимость, однако оно значительно ускоряет доступ к информации. Важно, что процесс кэширования полностью автоматизирован и прозрачен для пользователя – он не участвует в выборе, какие данные перемещать между уровнями памяти.

В системах с поддержкой кэширования запрос к оперативной памяти обрабатывается по следующему *алгоритму:*

1. *Проверка наличия данных в кэш-памяти.* Для этого просматривается содержимое кэша по полю «адрес в оперативной памяти», соответствующему запрашиваемому значению. Поскольку кэш не является адресуемым в традиционном смысле, поиск выполняется по содержимому.

2. *Если данные уже находятся в кэш-памяти,* они немедленно извлекаются и передаются в процессор.

3. *Если данных нет в кэше,* то соответствующая информация вместе с адресом копируется из оперативной памяти в кэш. После этого данные предоставляются процессору.

При этом может возникнуть ситуация, когда в кэше недостаточно места для новых данных. Тогда система находит наименее востребованные элементы и заменяет их. Если вытесняемые данные были изменены, они предварительно записываются обратно в оперативную память. Если изменения не производились, ячейка просто освобождается.

В целях повышения эффективности в кэш обычно загружается не один элемент, а целый блок данных. Это позволяет увеличить вероятность так называемого «попадания в кэш», то есть ситуации, когда необходимые данные уже имеются в кэш-памяти.

На практике коэффициент попадания в кэш достигает значения около 0,9, что означает, что 90% запросов удовлетворяются без обращения к более медленной памяти. Такая высокая эффективность объясняется свойствами временной и пространственной локальности данных.

• *Пространственная локальность* предполагает, что после обращения к определенному адресу с высокой вероятностью последуют обращения к соседним адресам.

• *Временная локальность* означает, что если недавно происходил доступ к определенному адресу, то в ближайшее время, скорее всего, произойдет повторный доступ к нему же.

Принцип кэширования применим не только к взаимодействию между кэш-памятью и оперативной памятью, но и между другими уровнями ЗУ. Например, между оперативной и внешней памятью – в этом случае кэш-функции выполняет буфер, расположенный в оперативной памяти, что позволяет ускорить доступ к данным, находящимся на внешних носителях.

РАЗДЕЛ 7. ФАЙЛОВАЯ СИСТЕМА

Тема 7.1. Имена файлов

Файловая система представляет собой компонент операционной системы, главная задача которого – организация удобного доступа пользователя к данным, размещённым на накопителях, а также обеспечение возможности одновременного использования файлов несколькими процессами и пользователями.

В более широком понимании термин «файловая система» охватывает следующие аспекты:

• совокупность всех файлов, находящихся на накопителе;

• структуры данных, применяемые для управления файлами, такие как каталоги, дескрипторы, таблицы размещения, фиксирующие свободное и занятое пространство на диске;

• комплекс программных инструментов системы, реализующих функциональность по работе с файлами, включая их создание, удаление, чтение, запись, поиск, переименование и прочие действия.

Именование файлов

Каждому файлу присваивается имя, которое служит его идентификатором в файловой системе. Пользователь может дать файлу символьное имя, при этом должны соблюдаться ограничения, накладываемые операционной системой на допустимые символы и максимальную длину имени. В прошлом эти ограничения были достаточно строгими. Например, в широко распространённой файловой системе FAT применялась схема 8.3 – имя файла состояло из восьми символов, за которыми следовало трёхсимвольное расширение. В UNIX System V длина имени ограничивалась 14 символами.

Однако такие короткие имена не всегда удобны, особенно при длительной работе с файлами. Длинные символьные имена, напротив, позволяют использовать легко запоминающиеся, осмысленные названия, отражающие содержимое файла. Благодаря этому современные файловые системы позволяют использовать значительно более длинные имена. Например, файловая система NTFS, применяемая в Windows NT, поддерживает имена длиной до 255 символов (без учёта завершающего нулевого символа).

Переход на длинные имена создал необходимость совместимости с устаревшими программами, которые продолжают работать с короткими именами. Чтобы такие программы могли получать доступ к новым файлам,

система должна уметь автоматически генерировать короткие имена (псевдонимы), соответствующие длинным. Это позволяет сохранить работоспособность старых приложений.

Поддержка длинных имён реализована как в современных файловых системах, так и в новых модификациях уже существующих. В качестве примера можно привести файловую систему VFAT, появившуюся в Windows 95 и представляющую собой усовершенствованную версию FAT. Одним из её ключевых преимуществ стала поддержка длинных имён. При этом важной задачей было обеспечить хранение таких имён без существенного изменения существующих структур на диске.

В ряде случаев возможно существование нескольких файлов с одинаковыми символьными именами. В подобных ситуациях уникальная идентификация файла достигается за счёт использования составного имени, представляющего собой иерархическую последовательность имён каталогов, ведущих к данному файлу. В некоторых операционных системах каждому файлу разрешено иметь только одно имя, тогда как в других допускается наличие нескольких имён для одного файла. Для обеспечения однозначности операционная система может присваивать каждому файлу внутреннее уникальное имя – числовой идентификатор, применяемый на уровне ОС. Например, в UNIX таким идентификатором служит номер индексного дескриптора (inode).

Типы файлов

В операционной системе файлы классифицируются на различные *типы.* Основными являются:

- *Обычные файлы;*
- *Специальные файлы;*
- *Каталоги.*

Обычные файлы делятся на текстовые и двоичные. Текстовые файлы представляют собой последовательности строк символов, закодированных в ASCII. Это могут быть документы, исходные коды программ и другие данные, читаемые пользователем. Такие файлы можно просматривать с помощью текстовых редакторов и выводить на печать.

Двоичные файлы, в отличие от текстовых, имеют более сложную структуру и не используют кодировку ASCII. Они могут содержать машинный код программ, архивы, образы дисков и другие данные, предназначенные для обработки программами. Операционная система должна как минимум уметь обрабатывать свои собственные исполняемые файлы, которые являются одним из видов двоичных

Специальные файлы – это файлы, которые ассоциированы с устройствами ввода и вывода. Они позволяют пользователю обращаться к периферийным устройствам с использованием обычных операций чтения и запи-

си. При обращении к такому файлу файловая система сначала обрабатывает запрос, а затем передаёт его на уровень операционной системы, где он преобразуется в команду для соответствующего оборудования. Специальные файлы подразделяются на:

• *Блок-ориентированные* (например, диски);

• *Байт-ориентированные* (например, последовательные порты).

Каталоги выполняют двойную функцию. С одной стороны, это логическое объединение файлов по смыслу (например, файлы игр или одного программного проекта), с другой – это структурированные файлы, содержащие сведения о входящих в них объектах. Внутри каталога хранится перечень файлов и соответствующих атрибутов, таких как имя, тип, размер и другие характеристики (рис. 11).

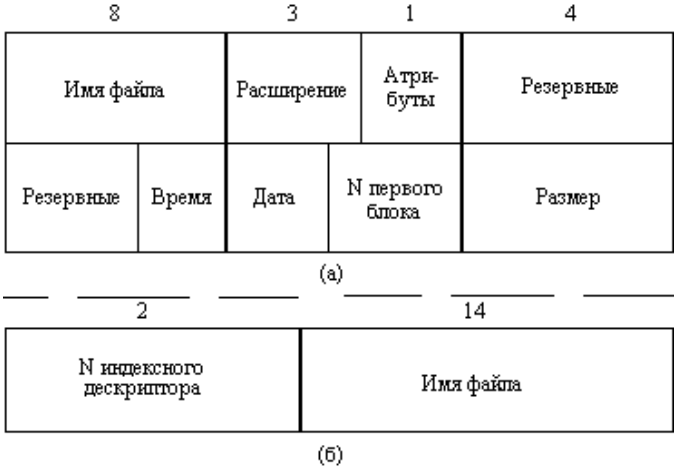


Рис. 11. Структура каталогов: а – структура записи каталога MS-DOS (32 байта); б – структура записи каталога ОС UNIX

В зависимости от типа файловой системы *набор атрибутов* может различаться. Среди них могут быть:

• данные об уровне доступа (права),

• имя владельца и создателя,

• флаг «только для чтения»,

• метки «скрытый», «системный», «архивный»,

• признак символьного или двоичного формата,

• флаг временного файла (удаляется после завершения работы процесса),

• блокировки,

• информация о длине записи и ключевых полях,

• временные метки (создание, изменение, последний доступ),

• текущий и максимальный объём файла.

Каталоги могут содержать эти атрибуты напрямую (как в MS-DOS), либо ссылаться на отдельные таблицы, где хранится информация (как в UNIX). Каталоги организуются в иерархическую структуру: один каталог может включать в себя другие, образуя многоуровневую систему (рис. 12).

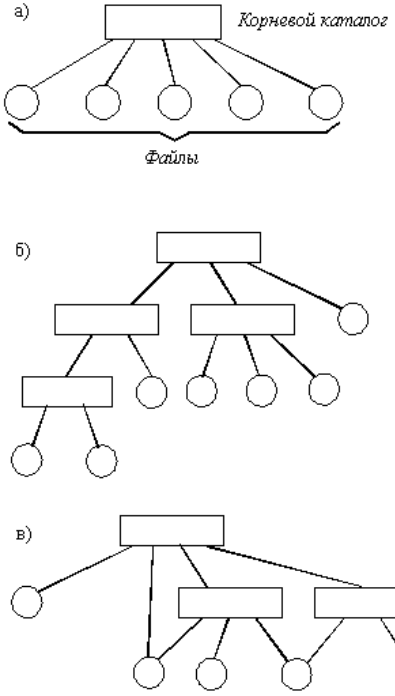


Рис. 12. Логическая организация файловой системы а – одноуровневая; б – иерархическая (дерево); в – иерархическая (сеть)

Иерархия каталогов

Файловые системы могут реализовывать иерархию в виде *дерева* или *сети.*

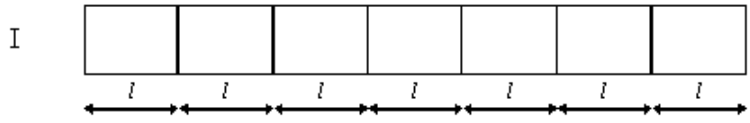
Если файл может принадлежать только одному каталогу, то структура является древовидной. Это характерно, например, для файловой системы MS-DOS. Если один и тот же файл может быть связан с несколькими каталогами, структура превращается в сеть, как в системе UNIX.

Каталоги, как и любые файлы, имеют символьные имена и идентифицируются с помощью составных имён, включающих последовательность имён всех вложенных каталогов от корневого до нужного.

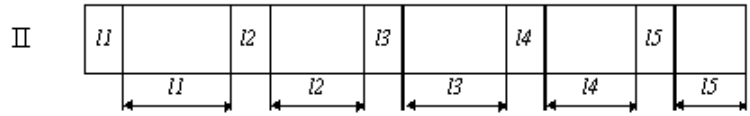
Тема 7.2. Физическая организация и адрес файла, права доступа к файлу

Логическая организация файла

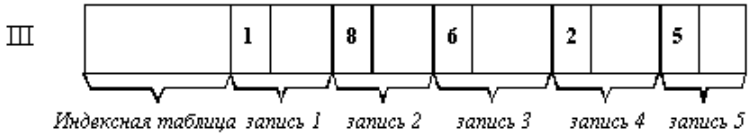
Программисты работают с файлами на уровне их логической организации (рис. 13), представляя файлы как набор логических записей. Логическая запись – это минимальная единица данных, с которой работает программа при взаимодействии с внешними устройствами. Физическое хранение данных может осуществляться большими блоками, однако операционная система предоставляет доступ к отдельным логическим записям. Логические записи могут быть как фиксированной, так и переменной длины, а расположение этих записей может быть последовательным или индексно-последовательным (с использованием таблиц индексов для быстрого доступа).



Последовательность логических записей фиксированной длины



Последовательность логических записей переменной длины



Индексная логическая организация

Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс ≡ ключ

Рис. 13. Способы логической организации файлов

Физическая организация и адрес файла

Физическая организация файла (рис. 14) касается того, как файл размещается на внешнем устройстве (например, на диске). Файл состоит из фи-

зических блоков – минимальных единиц данных, которые система передает между оперативной памятью и диском.

• *Непрерывное размещение:* Самый простой метод, при котором файлу выделяется последовательность блоков диска. Этот метод имеет два недостатка: заранее неизвестна длина файла, что может привести к неэффективному использованию пространства, и возможна фрагментация.

• *Связанный список блоков:* В этом способе каждый блок файла содержит указатель на следующий блок. Это помогает избежать фрагментации, но усложняет произвольный доступ к данным, поскольку для чтения данных необходимо последовательно пройти через все блоки.

• *Связанный список индексов:* В этом случае к каждому блоку привязывается индекс, который содержит указатель на следующий блок. Это позволяет быстрее находить нужные блоки и избегать проблем с фрагментацией.

• *Перечень номеров блоков:* Этот метод используется, например, в операционных системах UNIX, где адреса блоков файла хранятся в нескольких полях, что позволяет эффективно управлять большими файлами.

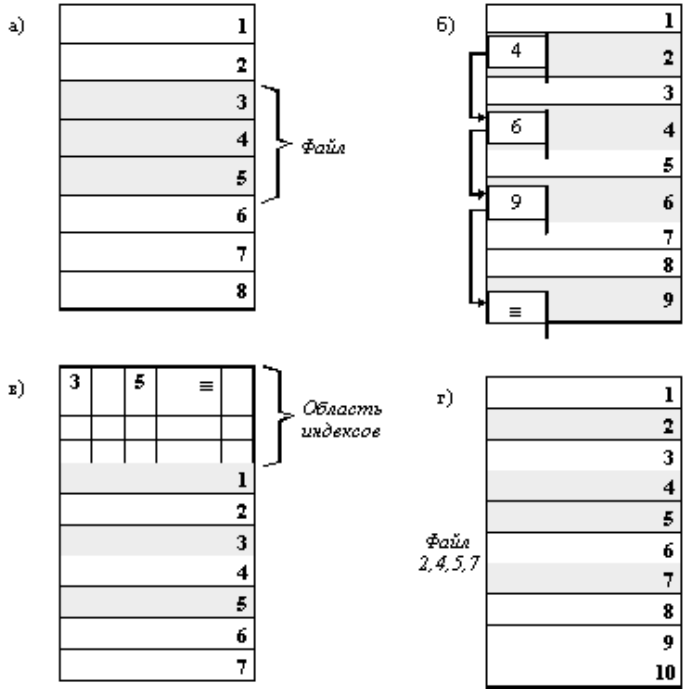


Рис. 14. Физическая организация файла а – непрерывное размещение; б – связанный список блоков; в – связанный список индексов; г – перечень номеров блоков

Права доступа к файлу

Права доступа к файлу регулируют операции, которые каждый пользователь может выполнять с файлом. Типичные операции включают создание, уничтожение, чтение, запись, а также изменение атрибутов файла и переименование (рис. 15).

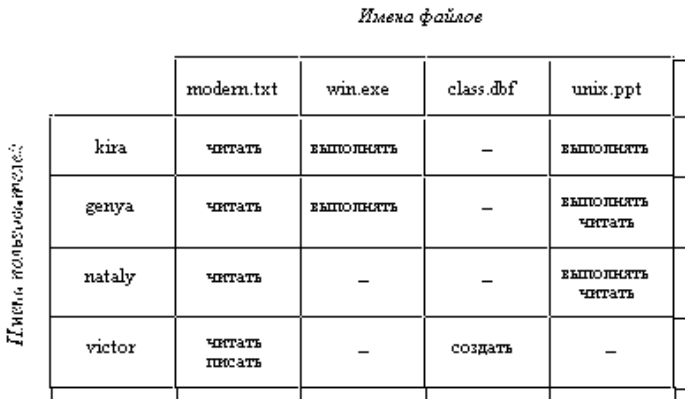


Рис. 15. Матрица прав доступа

Права могут быть настроены по различным *моделям:*

1. *Избирательный доступ:* Владелец файла может сам решать, какие операции разрешены для разных пользователей.

2. *Мандатный доступ:* Права доступа определяются системой в зависимости от категории пользователя (например, владельца, группы или остальных).

Для управления правами доступа может быть использована матрица прав доступа, где строки – это пользователи, а столбцы – файлы. В каждой ячейке указывается, какие операции доступны для соответствующего файла и пользователя.

Кэширование диска

В некоторых файловых системах используется система буферизации для ускорения работы с дисками. Подсистема буферизации сохраняет данные в оперативной памяти, что позволяет уменьшить количество операций с физическим диском. Когда данные запрашиваются, система сначала проверяет, есть ли нужный блок в памяти, и если он есть, предоставляет его без обращения к диску. Это уменьшает время доступа и улучшает производительность.

Общая модель файловой системы

Модель файловой системы (рис. 16) часто представляется многоуровневой, где каждый уровень системы предоставляет интерфейс для работы с данными и использует интерфейс нижележащего уровня.

Рис. 16. Общая модель файловой системы

Модель *включает:*

1. *Символьный уровень* – связывает символьное имя файла с уникальным идентификатором.

2. *Базовый уровень* – управляет характеристиками файла (размер, права доступа, адрес и т. д.).

3. *Проверка прав доступа* – проверяет разрешения для выполнения операций с файлом.

4. *Логический уровень* – определяет координаты логической записи в файле.

5. *Физический уровень* – определяет, какой физический блок содержит нужную запись.

Отображаемые в память файлы

Отображение файлов в адресное пространство (рис. 17) процесса является механизмом, который позволяет организовать доступ к файлам напрямую через память, значительно упрощая работу с данными. Этот подход был впервые использован в операционной системе MULTICS и с тех пор активно применяется в различных современных ОС. Основной идеей является использование двух системных вызовов: MAP (отображение) и UNMAP (отмена отображения). При использовании вызова MAP операционная система отображает файл в виртуальное адресное пространство процесса, что позволяет работать с данным файлом как с обычной памятью.

Примером может служить ситуация, когда файл размером 64 К отображается в виртуальное пространство, начиная с адреса 512 К. В таком случае чтение данных по адресу 512 К возвращает первый байт файла, а запись по адресу 512 К + 1100 изменяет соответствующий байт в файле. Такой подход позволяет модифицировать файлы через обычные операции с памятью, например, с использованием команды mov.

При этом операционная система использует механизм страничной адресации. При первом доступе к данным, страница файла загружается в физическую память, а изменения в файле фиксируются только при его вытеснении из памяти. Когда процесс завершает работу, все изменения переносятся в файл.

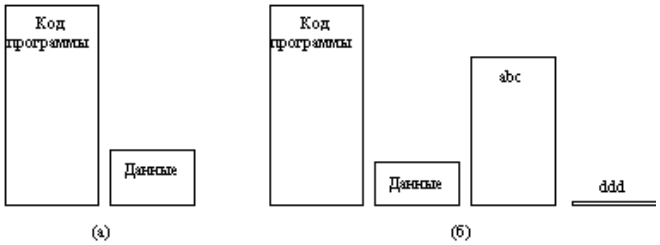


Рис. 17. (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) Процесс после отображения существующего файла abc в один сегмент и создания нового сегмента для файла ddd

Тем не менее, использование отображения файлов вызывает несколько проблем. Во-первых, система не может точно определить длину изменённого файла, так как она не отслеживает, сколько байт было изменено в пределах каждой страницы. Второй проблемой является синхронизация данных между процессами: если один процесс изменяет файл, а другой пытается его открыть для обычного доступа, изменения не будут видны до того момента, пока страница не будет вытеснена на диск. Третьей проблемой является возможность того, что файл может быть слишком большим для одного сегмента или даже для всего виртуального адресного пространства.

Современные архитектуры файловых систем

В современных операционных системах для работы с несколькими файловыми системами используется многоуровневая архитектура. Примером является Windows 95, в которой используется устанавливаемый диспетчер файловой системы (IFS). Этот компонент осуществляет интерфейс между запросами приложений и конкретной файловой системой, преобразуя их в формат, воспринимаемый системой.

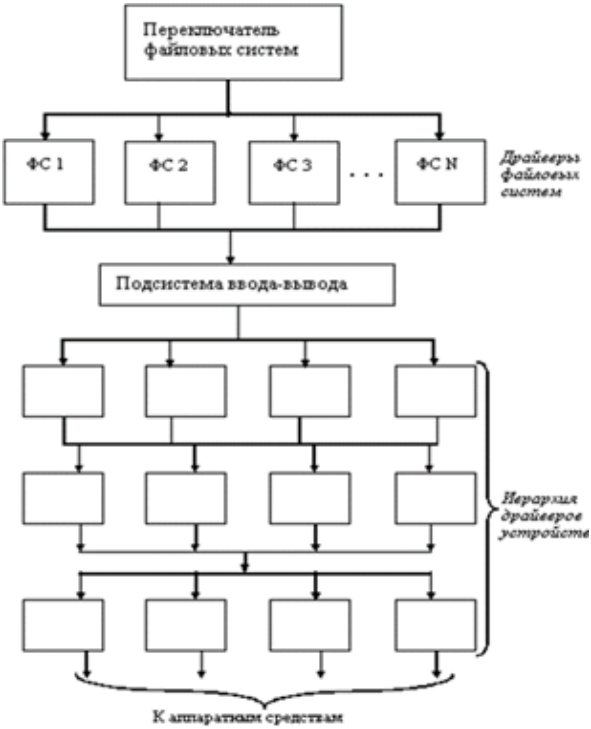


Рис. 18. Архитектура современной файловой системы

Архитектура файловой системы состоит из нескольких *слоёв* (рис. 18):

1. *Переключатель файловых систем* – обеспечивающий интерфейс между приложениями и файловыми системами.

2. *Драйверы файловых систем* – модули, реализующие конкретные типы файловых систем.

3. *Подсистема ввода-вывода* – управляет модулями, которые обеспечивают взаимодействие с физическими устройствами.

Такой подход позволяет достичь гибкости в работе с файлами и адаптировать систему под различные устройства и файловые системы, обеспечивая при этом высокую степень совместимости.

Тема 7.3. Сетевые операционные системы. Структура сетевой операционной системы

Понятие сетевой операционной системы. Компьютерная сеть

Сетевая операционная система (ОС) служит основой для работы в компьютерных сетях, предоставляя пользователям доступ к различным ресурсам, хранимым на других компьютерах. В отличие от ОС автономного компьютера, сетевая ОС обеспечивает взаимодействие между отдельными компьютерами в сети. Компьютерная сеть представляет собой набор компьютеров, соединённых с помощью коммуникационных систем, с программным обеспечением, которое позволяет пользователям использовать ресурсы, размещённые на других устройствах. Сетевая ОС выполняет роль интерфейса, скрывающего детали взаимодействия с низкоуровневыми аппаратными средствами сети.

Существует *два основных типа операционных систем,* работающих в сети:

• *Сетевые ОС* – это системы, которые позволяют компьютерам работать в сети, предоставляя ресурсы для совместного использования.

• *Распределённые ОС* – они скрывают распределённость системы и создают иллюзию работы с единой виртуальной машиной, что делает распределённые вычисления более прозрачными для пользователя.

Пользователь сетевой ОС всегда осведомлён о местоположении своих файлов и задач, в то время как распределённые ОС устраняют эту необходимость, предлагая пользователю единый виртуальный интерфейс. На практике большинство сетевых ОС ещё не обладают полноценной распределённостью.

Типичная структура сетевых операционных систем

Сетевые операционные системы включают несколько ключевых компонентов, которые обеспечивают управление ресурсами и их доступность через сеть:

• *Средства управления локальными ресурсами:* эти функции реализуют стандартные операции ОС автономного компьютера.

• *Сетевые средства:*

– *Серверная часть ОС* – предоставляет локальные ресурсы и услуги для использования другими компьютерами в сети.

– *Клиентская часть ОС* – отвечает за запросы доступа к удалённым ресурсам и услугам.

– *Транспортные средства ОС* – совместно с коммуникационной системой обеспечивают передачу сообщений между компьютерами, включая их формирование, разбиение на пакеты, преобразование имен в адреса и маршрутизацию.

Основные функции клиентской части

Клиентская часть ОС имеет несколько ключевых *функций:*

• Распознавание и перенаправление запросов к удалённым файлам.

• Преобразование форматов запросов, отправляемых приложениями, в формат, понятный серверной части ОС.

Сетевые службы и сетевые ресурсы

Сетевые службы представляют собой совокупность серверной и клиентской частей ОС, которые обеспечивают доступ к различным ресурсам через сеть. Например, служба файловой системы включает как сервер, так и клиент, которые обеспечивают доступ к данным на удалённом сервере.

Сетевые сервисы, в свою очередь, обеспечивают доступ к определённому типу ресурсов, таким как принтеры (служба печати), электронная почта (почтовая служба), или удалённый доступ к сети.

Сетевые службы могут быть ориентированы как на пользователя, так и на администратора:

• *Службы, ориентированные на пользователя:* обеспечивают доступ к ресурсам, как, например, файловые или почтовые службы.

• *Службы, ориентированные на администратора:* включают службы для мониторинга сети, резервного копирования и архивирования, службы каталогов и безопасности.

Архитектура взаимодействия «клиент – сервер»

Сетевые операционные системы часто используют архитектуру «клиент – сервер», в которой сервер предоставляет ресурсы и услуги, а клиент получает к ним доступ. Сервер может предоставлять данные, программы или другие ресурсы для общего пользования. Клиенты, в свою очередь, взаимодействуют с сервером для получения необходимых сервисов.

Преимущества архитектуры «клиент – сервер» включают централизованное управление, безопасность и скорость доступа. Администрирование сети позволяет поддерживать эффективность работы и безопасность всех подключённых устройств.

Таким образом, сетевая ОС выполняет важную роль в организации работы компьютерных сетей, обеспечивая доступ к ресурсам и организуя взаимодействие между компьютерами, а архитектура «клиент – сервер» помогает эффективно управлять этими ресурсами.

СПИСОК ЛИТЕРАТУРЫ

1. Батаев А. В., Налютин Н. Ю., Синицин С. В. Операционные системы и среды: учебник для студ. учреждений сред. проф. образования. – Издательский центр «Академия», 2014. – 272 с.

2. Бородин А. М. Основы анализа сложных систем: Курс лекций по методологии исследования и проектирования сложных систем. − Тирасполь: ПГУ, 2006. − 144 с.

3. Васильев Р. Б. и др. Управление развитием информационных систем: учебное пособие. − Москва: Горячая линия–Телеком, 2002. − 44 с.

4. Избачков Ю. С. и др. Информационные системы: учебник. − Москва, Санкт-Петербург: Питер, 2011. − 544 с.

5. Гостев И. М. Операционные системы, учебник и практикум для СПО. − Москва: Издательство Юрайт, 2019. − 164 с.

6. Киселев С. В., Алексахин С. В., Остроух А. В. Операционные системы: учебное пособие. − Москва: Академия, 2010. − 64 с.

7. Современные операционные системы: учебное пособие / С. В. Назаров, А. И. Широков. – 2-е изд. испр. и доп. – Москва: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2013. – 367 с.: ил., табл.

8. Партыка Т. Л., Попов И. И. Операционные системы, среды и оболочки, из. 4-е. − Москва: Форум, 2011.

9. Таненбаум Э. С. Современные операционные системы. – 4-е изд. − Санкт-Петербург, 2012. − 224 с.

10. Цилькер Б. Я., Орлов С. А. Организация ЭВМ и систем. − Санкт-Петербург: Питер, 2007. − 668 с.

Учебное издание

ОПЕРАЦИОННЫЕ СИСТЕМЫ И СРЕДЫ
Учебное пособие

Составители:
Ольга Михайловна Фурдуй
Татьяна Сергеевна Новакова
Елена Григорьевна Яковенко

Издается в авторской редакции
Компьютерная верстка: Ю. А. Запорожан

ИЛ № 06150. Сер. АЮ от 21.02.02.
Подписано в печать 14.11.2025. Формат 60×90/16.
Усл. печ. л. 4,8. Электронное издание. Заказ № 705.