Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Т.П. Злыднева

ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ

Проблемно-информационный курс

*Утверждено Редакционно-издательским советом университета
в качестве учебного пособия*

Издание 3-е

Магнитогорск
2021

УДК 004.45

Рецензенты:

Заместитель директора представительства
Челябинского государственного педагогического университета
в г. Магнитогорске,
кандидат технических наук, доцент
Л.Е. Смушкевич

Доцент кафедры прикладной математики и информационных технологий
Сибайского института (филиала)
ФГБОУ ВПО «Башкирский государственный университет»,
кандидат педагогических наук
И.С. Гумеров

Злыднева Т.П.
Введение в операционные системы. Проблемно-информационный курс [Электронный ресурс] : учебное пособие / Татьяна Павловна Злыднева ; ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова». – Изд. 3-е, подгот. по печ. изд. 2020 г. – Электрон. текстовые дан. (0,94 Мб). – Магнитогорск : ФГБОУ ВО «МГТУ им. Г.И. Носова», 2021. – 1 электрон. опт. диск (CD-R). – Систем. требования : IBM PC, любой, более l GHz ; 512 Мб RAM ; 10 Мб HDD ; MS Windows XP и выше ; Adobe Reader 8.0 и выше ; CD/DVD-ROM дисковод ; мышь. – Загл. с титул. экрана.
ISBN 978-5-9967-2083-5

В представленном учебном пособии предлагается использование проблемно-информационного метода преподавания: часть материала излагается в виде лекций, а часть рекомендуется изучить самостоятельно, в форме решения учебной проблемы. Оно включает в себя основные сведения об операционных системах и может быть использовано при подготовке к лекционным и практическим занятиям по дисциплине «Операционные системы».

Учебное пособие предназначено для студентов, обучающихся по направлению подготовки 01.03.02 «Прикладная математика и информатика», может быть полезно студентам других специальностей, аспирантам, преподавателям, всем, кто имеет дело с компьютерами и хочет знать, как устроены и функционируют современные операционные системы.

УДК 004.45

ISBN 978-5-9967-2083-5

© Злыднева Т.П., 2020
© ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова», 2020

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 5
ВВЕДЕНИЕ 6
1. ФУНКЦИИ ОПЕРАЦИОННЫХ СИСТЕМ 8
1.1. Основные понятия и назначение ОС 8
1.2. Функциональные компоненты ОС 12
1.2.1. Управление процессами 12
1.2.2. Управление памятью 14
1.2.3. Управление файлами и внешними устройствами 15
1.2.4. Защита данных и администрирование 16
1.2.5. Интерфейс прикладного программирования 17
1.2.6. Пользовательский интерфейс 19
1.3. Прочие функции операционной системы 20
1.4. Основные принципы построения ОС 21
Контрольные вопросы 22
2. ЭВОЛЮЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ (учебная проблема) 22
Контрольные вопросы 22
3. АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ 24
3.1. Модульная организация ОС 24
3.2. Привилегированный режим ядра ОС 26
3.3. Многослойная структура ОС 28
3.4. Аппаратная зависимость и переносимость ОС 31
3.4.1. Типовые средства аппаратной поддержки ОС 31
3.4.2. Машинно-зависимые компоненты ОС 32
3.4.3. Переносимость операционной системы 34
3.5. Микроядерная архитектура 36
3.6. Совместимость и множественные прикладные среды 39
Контрольные вопросы 42
4. КЛАССИФИКАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ
(учебная проблема) 43
Контрольные вопросы 43
5. ПРОЦЕССЫ И ПОТОКИ 44
5.1. Мультипрограммирование 44
5.1.1. Мультипрограммирование в системах пакетной обработки 44
5.1.2. Мультипрограммирование в системах разделения времени . 46
5.1.3. Мультипрограммирование в системах реального времени ... 47
5.2. Мультипроцессорная обработка 48
5.3. Планирование процессов и потоков 51
5.3.1. Создание процессов и потоков 51
5.3.2. Планирование и диспетчеризация потоков 54

5.3.3. Алгоритмы планирования потоков 57
5.4. Прерывания 63
5.4.1. Назначение и типы прерываний 63
5.4.2. Механизм прерываний 65
Контрольные вопросы 67
6. ОПЕРАЦИОННАЯ СИСТЕМА И ПАМЯТЬ 68
6.1. Типы адресации 68
6.2. Алгоритмы распределения памяти 70
6.2.1. Распределение памяти разделами 70
6.2.2. Страничное распределение памяти 73
6.2.3. Сегментное распределение памяти 75
6.2.4. Сегментно-страничное распределение памяти 77
6.3. Кэш-память 79
Контрольные вопросы 82
7. СОВРЕМЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ 83
7.1. Операционные системы семейства Windows 83
7.1.1. Основные характеристики 83
7.1.2. Служебные приложения 85
7.1.3. Настройка операционной системы 88
7.1.4. Эволюция ОС семейства Windows (учебная проблема) 90
Контрольные вопросы 90
7.2. Операционные системы семейства UNIX (учебная проблема) 91
Контрольные вопросы 91
7.3. Сетевые операционные системы (учебная проблема) 92
Контрольные вопросы 92
7.4. Требования к современным операционным системам 94
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 97

ПРЕДИСЛОВИЕ

В настоящее время возросла потребность в специалистах, способных самостоятельно ориентироваться в потоке меняющейся информации, использующих в своей работе все то новое, что появляется в области информатики и вычислительной техники, способных сравнивать, анализировать, находить лучшие варианты решений, то есть проводить исследования. Выпускник высшего учебного заведения в профессиональной деятельности сможет постоянно совершенствовать свою квалификацию, если за время обучения в вузе получил навык самостоятельной, исследовательской работы, освоил ее методы, научился их применять при изучении конкретных общепрофессиональных дисциплин.

Решением вопросов организации исследовательской деятельности студентов университета в процессе профессиональной подготовки мы занимаемся достаточно давно. Особое внимание в данном направлении уделено организации учебно-исследовательской деятельности студентов при изучении дисциплин информатики [10 13, 16, 26–27]. Представленное учебное пособие помимо классического изложения теории об операционных системах предусматривает использование исследовательского метода, что способствует интенсификации познавательной активности обучающихся. Именно при изучении операционных систем мы предлагаем организацию учебно-исследовательской деятельности студентов. Формой организации и управления этой деятельностью служит проблемное обучение. Учебный курс построен таким образом, что в нем выделена совокупность теоретически и практически значимых задач, предоставлена возможность самостоятельного разрешения студентом поставленной проблемной ситуации. При этом часть материала изучается обычными репродуктивными методами. В учебном пособии разделы, которые предполагают самостоятельное познание новых теоретических фактов, овладение методами поиска выполнения заданий, отмечены как «учебная проблема». Сочетание исследовательского и репродуктивного методов при изучении операционных систем способствует тому, чтобы исследовательская деятельность студентов стала органической частью учебного процесса и имела профессиональную направленность.

В последние годы, при реализации ФГОС ВПО, главной целью на различных ступенях высшего образования является формирование у студентов в ходе обучения компетенций. Опыт нашей работы показал, что грамотный выбор средств, форм, методов обучения, используемых для формирования компетенций, значительно помогает решить проблему качества подготовки, как бакалавров, так и магистров, и именно учебно-исследовательская деятельность играет в этом значительную роль [22–25].

ВВЕДЕНИЕ

Дисциплина «Операционные системы» изучается бакалаврами направления подготовки 01.03.02 «Прикладная математика и информатика» в пятом семестре. На данном этапе подготовки необходимо сформировать первоначальные навыки исследовательской деятельности студентов в различных ее формах. Предлагаемая методика ориентирована на лекции проблемно-информационного характера, написание рефератов и семинарские занятия исследовательского типа.

Проблемно-информационный метод преподавания предполагает деятельность педагога по организации решения совместно с обучающимися учебных проблем. При этом необходимо оптимально сочетать на отдельных этапах разъяснение, сообщение учебной информации.

В силу того, что дисциплина «Операционные системы» является узкопрофессиональной и знаний в этой области у студентов мало, значительная часть материала должна быть изучена обычным репродуктивным методом (получение информации – воспроизведение ее). Таким способом предполагается изучение разделов «Функции операционных систем», «Архитектура операционных систем», «Процессы и потоки», «Операционная система и память», частично «Современные операционные системы». Вопросы к каждому из разделов позволяют провести контроль полученных знаний.

Другая часть материала – «Эволюция операционных систем», «Классификация операционных систем», «Эволюция ОС семейства Windows», «Операционные системы семейства UNIX», «Сетевые операционные системы» – изучается исследовательским методом, в учебном пособии она отмечена как «учебная проблема». Начав с создания познавательной потребности в решении возникшей в результате постановки проблемной ситуации, необходимо добиться осознания студентами проблемы, провести поиск гипотезы, касающейся результата и пути его получения. Все это проводится на лекционном занятии, следовательно, каждый студент оказывается вовлеченным в учебно-исследовательскую деятельность.

Границы применения исследовательского метода определяются фактором времени: с одной стороны, на усвоение материала требуется больше времени, а с другой, в условиях лекционных занятий пределы применения этого метода сведены к минимуму. Поэтому решение проблемы, которое является основой перехода к следующей учебной проблеме и ведет к открытию нового знания, предполагается проводить вне лекционных часов.

По тем разделам, которые помечены как «учебная проблема», предлагается написать рефераты, распределив их между студентами или

микрогруппами студентов. При этом преподаватель не «выдает» тему, а только формулирует учебную проблему, студенты самостоятельно выбирают тему реферата для решения поставленной проблемы. Преподаватель не рекомендует литературу, предоставляя студентам полную свободу выбора источника информации, и содержание работы оценивается не только преподавателем, так как планируется публичное выступление на семинаре. Все это оказывает влияние на оригинальность выполнения задания, личное отношение к проблеме. Рефераты предполагается распределить таким образом, чтобы в каждой подгруппе на семинарском занятии, в обязательном порядке была предоставлена возможность выступить как минимум двум студентам по одной и той же проблеме. В этом случае решением поставленных на лекциях учебных проблем будет занято максимальное количество учащихся.

Таким образом, выполнение письменного задания (реферата) способствует получению навыков самостоятельной работы, связанной с поиском, изучением и анализом научной литературы, развитию умений по оформлению результатов исследования, получению первоначального опыта публичного выступления.

Обмен информацией, полученной студентами в ходе самостоятельного исследования по поставленной проблеме, рекомендуется организовать в рамках семинарских занятий. Здесь предполагается выступление студентов с рефератами, их оценка, обсуждение, обобщение отдельных фактов и событий, решение учебной проблемы в целом. Ценность данной формы занятий в том, что в процессе обсуждения можно высказать собственное мнение и попытаться доказать его правильность.

В учебном пособии для каждого раздела, помеченного как «учебная проблема», предлагается большой перечень контрольных вопросов. Возможны три варианта использования данных вопросов при изучении теоретического материала: либо для контроля полученных студентами знаний по окончании семинара, либо для обсуждения каждого вопроса как мини-проблемы в ходе семинарского занятия, либо то и другое в определенном сочетании. Семинар исследовательского типа не только способствует углубленной проработке теоретического материала предмета на протяжении всего изучения курса, но и развивает творческую самостоятельность студентов, способность к обобщениям, содействует выработке практических навыков работы.

Предлагаемая методика успешно используется нами для изучения других дисциплин на различных ступенях высшего образования – и на бакалавриате, и в магистратуре [28–39].

1. ФУНКЦИИ ОПЕРАЦИОННЫХ СИСТЕМ

1.1. Основные понятия и назначение ОС

Одна из главных составных частей системного программного обеспечения – операционная система (ОС). Она представляет собой организованную совокупность программ, производит диалог с пользователем, осуществляет управление компьютером, его ресурсами (например, оперативной памятью и т.д.). Операционная система создает среду для выполнения прикладных программ и во многом определяет, какими полезными для пользователя свойствами эти программы будут обладать.

Основная причина необходимости ОС состоит в том, что элементарны операции для работы с устройствами компьютера и управления ресурсами компьютера – это операции очень низкого уровня, поэтому действия, которые необходимы пользователю и прикладным программистам, состоят из нескольких сотен или тысяч таких элементарных операций. ОС скрывает от пользователя эти сложные и ненужные подробности и предоставляет ему удобный интерфейс для работы.

Сегодня существует большое количество разных типов ОС, отличающихся областями применения, аппаратными платформами и методами реализации, что обуславливает и значительны функциональные различия этих ОС. Поэтому при изучении операционных систем очень важно из всего многообразия выделить те функции, которые присущи всем ОС.

Операционная система выполняет две группы функций:

1. Предоставление пользователю или программисту вместо реальной аппаратуры компьютера расширенной виртуальной машины, с которой легче работать и которую легче программировать.

2. Повышение эффективности использование компьютера путем рационального управления его ресурсами.

ОС как *виртуальная машина* реализует следующие функции:

а) управление информацией – структурирование, обеспечение сохранности, использование имён (виртуальная память, файлы), передача;

б) выполнение: последовательное или параллельное выполнение программ, компоновка программ и т. д.;

в) дополнительные услуги: помощь при отладке, обработка аварийных ситуаций, измерение времени выполнения и т. д.

Концепция виртуальности при построении ОС имеет большое значение. «Виртуальный» – мнимый. «Виртуальный ресурс» – некоторая модель физического ресурса. Если дать точное определение виртуальной машины, то оно звучит так:

Виртуальная машина – функциональный эквивалент некоторой воображаемой ЭВМ с заданной конфигурацией, моделируемой программно-аппаратными средствами конкретной реальной ЭВМ. Виртуальная память – оперативная память виртуальной машины.

Виртуальная память – это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память.

Для этого виртуальная память решает следующие задачи:

– размещает данные в запоминающих устройствах разного типа (например, часть программы в оперативной памяти, часть на диске);

– перемещает по мере необходимости данные между запоминающими устройствами разного типа (например, подгружает нужную часть программы с диска в оперативную память);

– преобразует виртуальные адреса в физические.

Если бы программист работал непосредственно с аппаратурой компьютера, без участия ОС, то, например, для организации чтения блока данных с диска программисту пришлось бы использовать более десятка команд с указанием множества параметров: номера блока на диске, номера сектора на дорожке и т.п. После завершения операции обмена с диском он должен был бы предусмотреть в своей программе анализ результата выполнения операции. Учитывая, что контроллер диска способен распознать более двадцати различных вариантов завершения операции, можно считать программирование обмена с диском на уровне аппаратуры не самой простой задачей. Не менее обременительной выглядит и работа пользователя, если бы ему для чтения файла с терминала потребовалось задавать числовые адреса дорожек и секторов.

Операционная система избавляет программистов не только от необходимости напрямую работать с аппаратурой дискового накопителя, предоставляя им простой файловый интерфейс, но и берёт на себя все другие рутинные операции, связанные с управлением другими аппаратными устройствами компьютера: физической памятью, таймерами, принтерами и т.д.

В результате реальная машина, способная выполнять только небольшой набор элементарных действий, определяемых её системой команд, превращается в виртуальную машину, выполняющую широкий набор гораздо более мощных функций. Виртуальная машина тоже управляется командами, но это уже команды другого, более высокого уровня: удалить файл с определенным именем, запустить на выполнение некоторую прикладную программу, повысить приоритет задачи, вывести текст из файла на печать и др.

Таким образом, назначение ОС состоит в предоставлении пользователю или программисту некоторой расширенной виртуальной машины,

которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальный компьютер.

Виртуализация памяти может быть осуществлена на основе двух различных подходов:

1. Свопинг – образы процессов выгружаются на диск и возвращаются в оперативную память *целиком*.

2. Виртуальная память – между оперативной памятью и диском перемещаются *части* (страницы, сегменты) образов процессов.

Свопинг является частным случаем виртуальной памяти, это более простой в реализации способ совместного использования оперативной памяти и диска. Свопингу свойственно перемещение избыточной информации, что замедляет работу системы и приводит к неэффективному использованию памяти. Кроме того, системы, поддерживающие свопинг, имеют еще один очень существенный недостаток: они не способны загрузить для выполнения процесс, виртуальное адресное пространство которого превышает имеющуюся в наличие свободную память. Именно эти недостатки стали причиной того, что свопинг как основной механизм управления памятью почти не используется в современных ОС. Его сменил более совершенный механизм *виртуальной памяти*.

В результате многократного изменения местоположения в оперативной памяти образов процессов или их частей, основной задачей виртуальной памяти является преобразование виртуальных адресов в физические. Решение этой задачи зависит от того, какой способ структуризации виртуального адресного пространства принят в данной системе управления памятью.

В настоящее время все множество реализаций виртуальной памяти может быть представлено тремя видами:

1. *Страничная виртуальная память* организует перемещение данных между памятью и диском страницами – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера (см. п.6.2.2. «Страничное распределение памяти»).

2. *Сегментная виртуальная память* предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных (см п.6.2.3. «Сегментное распределение памяти»).

3. *Сегментно-страничная виртуальная память* использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных здесь является страница (см. п.6.2.4. «Сегментно-страничное распределение памяти»). Этот способ управления памятью объединяет в себе элементы двух предыдущих подходов.

Другим назначением операционной системы является повышение эффективности использование компьютера путем рационального *управления его ресурсами*.

К числу основных ресурсов современных вычислительных систем могут быть отнесены такие ресурсы, как процессоры, основная память, таймеры, наборы данных, диски, накопители на магнитных лентах, принтеры, сетевые устройства и некоторые другие. Ресурсы распределяются между процессами.

Процесс (задача) представляет собой базовое понятие большинства современных ОС и часто кратко определяется как программа в стадии выполнения. Программа – это статический объект, представляющий собой файл с кодами и данными. Процесс – это динамический объект, который возникает в ОС после того, как пользователь или сама ОС решает «запустить программу на выполнение», т.е. создать новую единицу вычислительной работы (см. п.5 «Процессы и потоки»).

Дадим точное определение процесса (из толкового словаря программиста):

«Процесс – хронологически упорядоченная последовательность состояний некоторой системы. Для ОС процесс – единица работы, заявка на потребление системных ресурсов».

Критерий эффективности, в соответствии с которым ОС организует управление ресурсами компьютера, может быть различным. Например, в одних системах важен такой критерий, как пропускная способность вычислительной системы, в других – время её реакции. Соответственно выбранному критерию эффективности ОС по-разному организуют вычислительный процесс.

Управление ресурсами включает решение следующих задач:

а) планирование ресурса – определение, какому процессу, когда и в каком количестве (если ресурс может выделяться частями) следует выделить данный ресурс;

б) удовлетворение запросов на ресурсы;

в) отслеживание состояния и учет использования ресурса – поддерживание оперативной информации о том, занят или свободен ресурс и какая доля ресурса уже распределена;

г) разрешение конфликта между процессами.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, особенности которых, в конечном счете, и определяют облик ОС в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Например, применяемый алгоритм управления процессором в значительной степени определяет, может ли ОС использоваться как система

разделения времени, система пакетной обработки или система реального времени.

В отличие от функций расширенной (виртуальной) машины большинство функций управления ресурсами выполняются операционной системой автоматически и прикладному программисту недоступны.

1.2. Функциональные компоненты ОС

Функции ОС группируются либо в соответствии с типами локальных ресурсов, которыми управляет ОС, либо в соответствии со специфическими задачами, применимыми ко всем ресурсам. Такие группы функций называют подсистемами.

1. Подсистемы управления ресурсами:
– управление процессами;
– управление памятью;
– управление файлами и внешними устройствами.
2. Подсистемы, общие для всех ресурсов:
– защита данных и администрирование;
– прикладной интерфейс;
– пользовательский интерфейс.

Подробнее рассмотрим каждую подсистему.

1.2.1. Управление процессами

Для каждого вновь создаваемого процесса ОС генерирует системные информационные структуры, которые содержат данные о потребностях процесса в ресурсах вычислительной системы, а также о фактически выделенных ему ресурсах. Таким образом, процесс можно определить как некоторую заявку на потребление системных ресурсов (см. п.5 «Процессы и потоки»).

Чтобы процесс мог быть выполнен, ОС должна назначать ему область оперативной памяти, в которой будут размещены коды и данные процесса, а также предоставить ему необходимое количество процессорного времени.

В информационные структуры процесса часто включаются вспомогательные данные, характеризующие историю пребывания процесса в системе (например, какую долю времени процесс потратил на операции ввода/вывода, а какую на вычисления), его текущее состояние (активное или заблокированное), степень привилегированности процесса (значение приоритета). Данные такого рода могут учитываться операционной системой при принятии решении о предоставлении ресурсов процессу.

В ОС одновременно может существовать несколько процессов. Часть процессов порождается по инициативе пользователей и их прило-

жений, такие процессы обычно называют *пользовательскими*. Другие процессы, называемые *системными*, инициализируются самой ОС для выполнения своих функций.

Поскольку процессы часто одновременно претендуют на одни и те же ресурсы, то в обязанности ОС входит поддержание очереди заявок процессов на ресурсы, например очереди к процессору, к принтеру, к последовательному порту.

Важной задачей ОС является защита ресурсов, выделенных данному процессу, от остальных процессов. Одним из наиболее тщательно защищаемых ресурсов процесса являются области оперативной памяти, в которой хранятся коды и данные процесса.

Совокупность всех областей оперативной памяти, выделенных операционной системой процессу, называется его *адресным пространством*. Если говорят, что каждый процесс работает в своем адресном пространстве, то имеют в виду защиту адресных пространств, осуществляемую ОС.

На протяжении периода существования процесса его выполнение может быть многократно прервано и продолжено. Для того чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды:

– состояние регистров и программного счетчика;

– режим работы процессора;

– указатели на открытые файлы;

– информацию о незавершенных операциях ввода-вывода;

– коды ошибок, выполняемых данным процессом системных вызовов и т.д.

Эта информация называется *контекстом процесса*. Говорят, что при смене процесса происходит переключение контекстов.

В ОС нет достаточного соответствия между процессами и программами. Один и тот же программный файл может породить несколько параллельно выполняемых процессов, а процесс может в ходе своего выполнения сменить программный файл и начать выполнять другую программу.

Таким образом, подсистема управления процессами выполняет следующие функции:

1) планирует выполнение процессов (распределяет процессорное время между несколькими одновременно существующими в системе процессами);

2) занимается созданием и уничтожением процессов;

3) обеспечивает процессы необходимыми системными ресурсами;

4) поддерживает синхронизацию процессов;

5) обеспечивает взаимодействие между процессами.

1.2.2. Управление памятью

Память является для процесса таким же важным ресурсом, как и процессор, так как процесс может выполняться процессором только в том случае, если его коды и данные находятся в оперативной памяти.

Управление памятью включает в себя: распределение имеющейся физической памяти между всеми существующими в системе в данный момент процессами; загрузку кодов и данных процессов в отведенные им области памяти; настройку адресно-зависимых частей кодов процесса на физические адреса выделенной области; защиту областей памяти каждого процесса.

Существует множество алгоритмов распределения памяти. В некоторых системах распределение памяти выполняется страницами фиксированного размера, в других сегментами переменной длины (см. п.6 «Операционная система и память»). Одним из наиболее популярных способов управления памятью в современных операционных системах является так называемая *виртуальная память* (см. п.1.1.«Основные понятия и назначение ОС»). Все данные, используемые программой, хранятся на диске и при необходимости частями отображаются в физическую память. При перемещении кодов и данных между оперативной памятью и диском подсистема виртуальной памяти выполняет трансляцию виртуальных адресов, полученных в результате компиляции и компоновки программы, в физические адреса ячеек оперативной памяти.

Защита памяти – это избирательная способность предохранять выполняемую задачу от записи или чтения памяти, назначенной другой задаче.

Правильно написанные программы не пытаются обращаться к памяти, назначенной другим. Однако реальные программы часто содержат ошибки, в результате которых такие попытки иногда предпринимаются. Средства защиты памяти, реализованные в ОС, должны пресекать доступ процессов к чужим областям памяти.

Таким образом, функциями ОС по управлению памятью являются:

1) отслеживание свободной и занятой памяти;

2) выделение памяти процессам и освобождение памяти при завершении процессов;

3) защита памяти;

4) вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти недостаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;

5) настройка адресов программы на конкретную область физической памяти.

1.2.3. Управление файлами и внешними устройствами

Способность ОС к «экранированию» сложностей реальной аппаратуры очень ярко проявляется в одной из основных подсистем ОС – *файловой системе*.

Операционная система виртуализирует определенный набор данных, хранящихся на внешнем накопителе, в виде *файла* – простой неструктурированной последовательности байтов, имеющей символьное имя. Для удобства работы с данными файлы группируются в каталоги, которые в свою очередь, образуют группы – каталоги более высокого уровня.

Чтобы представить большое количество наборов данных, разбросанных случайным образом по цилиндрам и поверхностям дисков различных типов, в виде хорошо всем знакомой и удобной иерархической структуры файлов и каталогов, ОС должна решить множество задач. Файловая система ОС выполняет преобразование символьных имен файлов, с которыми работает пользователь или прикладной программист, в физические адреса данных на диске, организует совместный доступ к файлам, защищает их от несанкционированного доступа.

При выполнении своих функций файловая система тесно взаимодействует с подсистемой управления внешними устройствами, которая по запросам файловой системы осуществляет передачу данных между дисками и оперативной памятью.

Подсистема управления внешними устройствами, называемая также подсистемой ввода-вывода, исполняет роль интерфейса ко всем устройствам, подключенным к компьютеру (накопители на жестких, гибких и оптических дисках, принтеры, сканеры, мониторы, модемы, сетевые адаптеры, различные аналого-цифровые преобразователи и т.д.).

Программа, управляющая конкретной моделью внешнего устройства и учитывающая все его особенности, обычно называется *драйвером* этого устройства (от английского drive – вести, управлять). Драйвер может управлять единственной моделью устройства или же группой устройств определенного типа. Для пользователя важно, чтобы ОС включала как можно больше разнообразных драйверов, так как это гарантирует возможность подключения к компьютеру большего числа внешних устройств различных производителей. От наличия подходящих драйверов во многом зависит успех ОС на рынке (например, отсутствие многих необходимых драйверов внешних устройств было одной из причин низкой популярности OS/2). Созданием драйверов устройств занимаются как разработчики конкретной операционной системы, так и специалисты компаний, выпускающих внешние устройства.

Таким образом, одна из наиболее важных задач ОС – это поддержание высокоуровневого интерфейса прикладного программирования к разнообразным устройствам ввода-вывода.

Со времени появления ОС UNIX такой интерфейс в большинстве операционных систем строится на основе концепции файлового доступа. Эта концепция заключается в том, что обмен с любым внешним устройством выглядит как обмен с файлом, имеющим имя и представляющим собой неструктурированную последовательность байтов. В качестве файла может выступать как реальный файл на диске, так и алфавитно-цифровой терминал, печатающее устройство или сетевой адаптер.

1.2.4. Защита данных и администрирование

Безопасность данных вычислительной системы обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок программного обеспечения, а также средствами защиты от несанкционированного доступа. В последнем случае ОС защищает данные от ошибочного или злонамеренного поведения пользователей системы.

Первым рубежом обороны при защите данных от несанкционированного доступа является *процедура логического ввода*. Операционная система должна убедиться, что в системы пытается войти пользователь, вход которого разрешен администратором. Функции защиты ОС вообще очень тесно связаны с функциями администрирования, так как именно администратор определяет права пользователей при их обращении к разным ресурсам системы – файлам, каталогам, принтерам, сканерам и т.д. Кроме того, администратор ограничивает возможности пользователей в выполнении тех или иных системных действий (например, завершать чужие процессы).

Поддержка отказоустойчивости реализуется операционной системой на основе *резервирования*. Чаще всего в функции ОС входит поддержание нескольких копий данных на разных дисках. Резервируются так же принтеры и другие устройства ввода-вывода. При отказе одного из устройств операционная система должна быстро и прозрачно для пользователя образом провести реконфигурацию системы и продолжить работу с резервным устройством.

Поддержка отказоустойчивости также входит в обязанности системного администратора. В состав операционной системы обычно входят утилиты, позволяющие администратору выполнять регулярные операции резервного копирования для обеспечения быстрого восстановления важных данных.

1.2.5. Интерфейс прикладного программирования

Прикладные программисты используют в своих приложениях обращения к операционной системе, когда для выполнения тех или иных действий им требуется особый статус, которым обладает только ОС. Например, в большинстве современных ОС все действия, связанные с управлением аппаратными средствами компьютера, может выполнять только операционная система.

Помимо этих функций прикладной программист может воспользоваться набором сервисных функций ОС, которые упрощают написание приложений. Функции такого типа реализуют универсальные действия, часто требующиеся в различных приложениях, такие, например, как обработка текстовых строк. Эти функции могли бы быть выполнены и самим приложением, но гораздо проще использовать уже готовые, отлаженные процедуры, включенные в состав ОС. В то же время, даже при наличии в ОС соответствующей функции, программист может реализовать ее самостоятельно в рамках приложения, если предложенный операционной системой вариант его не вполне устраивает.

Возможности операционной системы доступны прикладному программисту в виде набора функций, называющегося *интерфейсом прикладного программирования* (Application Programming Interface, API). От конечного пользователя эти функции скрыты за оболочкой алфавитно-цифрового или графического пользовательского интерфейса.

Для разработчиков приложений все особенности конкретной ОС представлены особенностями ее API. Поэтому операционные системы с различной внутренней организацией, но с одинаковым набором функций API кажутся им одной и той же ОС, что упрощает стандартизацию операционных систем и обеспечивает переносимость приложений между внутренне различными операционными системами, соответствующими определенному стандарту на API.

Приложения выполняют обращения к функциям с помощью *системных вызовов*. Способ, которым приложение получает услуги ОС, очень похож на вызов подпрограмм. Информация, нужная ОС и состоящая обычно из идентификатора команды и данных, помещается в определенное место памяти, в регистры или стек. Затем управление передается операционной системе, которая выполняет требуемую функцию и возвращает результаты через память, регистры или стеки. Если операция проведена неуспешно, то результат включает индикацию ошибки.

Способ реализации системных вызовов зависит от структурной организации ОС, которая в свою очередь, тесно связана с особенностями аппаратной платформы. Кроме того, он зависит от языка программирования (ассемблер или языки высокого уровня).

От того, как ОС управляет работой приложений, во многом зависит надежность всей вычислительной системы. ОС должна предоставлять возможность прерывания работы приложений по желанию пользователя и снятия сбойной задачи без ущерба для работы других приложений. При этом требование надежности ОС может входить в противоречие с требованием ее универсальности. Например, наиболее универсальные ОС Windows 95 и Windows 98 могут испытывать общесистемные сбой из-за работы с приложениями, недостаточно четко соблюдающими спецификацию операционной системы. Операционные системы Windows NT и OS/2 обладают повышенной устойчивостью и не выходят из строя при сбое приложений, но имеют меньшую универсальность, и соответственно, количество доступных приложений для них ограниченно.

Установка приложений

Для правильной работы приложений на компьютере они должны пройти операцию, называемую *установкой*. Необходимость установки связана с тем, что разработчики программного обеспечения не могут заранее предвидеть особенности аппаратной и программной конфигурации вычислительной системы, на которой предстоит работать их программам.

Таким образом, *дистрибутивный комплект* (установочный пакет) программного обеспечения, как правило, представляет собой не законченный программный продукт, а полуфабрикат, из которого в процессе установки на компьютере формируется полноценное рабочее приложение. При этом осуществляется привязка приложения к существующей аппаратно-программной среде и его настройка на работу именно в этой среде.

Устаревшие операционные системы (например, MS-DOS) не имеют средств для управления установкой приложений. Единственное средство, которое они предоставляют, - возможность запуска устанавливающей программы, прилагаемой к дистрибутивному комплекту. Такая установка отличается крайней простотой, но и невысокой надежностью.

Современные графические ОС берут на себя управление установкой приложений. Они управляют распределением ресурсов вычислительной системы между приложениями, обеспечивают доступ устанавливаемых приложений к драйверам устройств вычислительной системы, формируют общие ресурсы, которые могут использоваться разными приложениями.

Удаление приложений

Процесс удаления приложений, как и процесс установки, имеет свои особенности и может происходить под управлением вычислительной системы. В таких ОС, где каждое приложение самообеспечено собственными ресурсами (например, в MS-DOS), его удаление не требует специального вмешательства операционной системы. Для этого доста-

точно удалить каталог, в котором размещается приложение, со всем его содержимым.

В операционных системах, реализующих принцип совместного использования ресурсов (например, в Windows), процесс удаления приложений имеет особенности. Нельзя допустить, чтобы при удалении одного приложения были удалены ресурсы, на которые опираются другие приложения, даже если эти ресурсы были когда-то установлены вместе с удаляемым приложением. В связи с этим удаление приложений происходит под строгим контролем операционной системы. Полнота удаления и надежность последующего функционирования ОС и оставшихся приложений во многом зависят от корректности установки и регистрации приложений.

1.2.6. Пользовательский интерфейс

Операционная система должна обеспечивать удобный интерфейс не только для прикладных программ, но и для человека, работающего за компьютером.

По реализации интерфейса пользователя различают *неграфические* (алфавитно-цифровые) и *графические* ОС.

Неграфические операционные системы организуют интерфейс *командной строки.* Основным устройством управления в данном случае является клавиатура. Управляющие команды вводят в поле командной строки, где их можно и редактировать. Исполнение команды начинается после ее утверждения, например нажатием клавиши Enter. Команды могут вводиться не только с терминала, но и считываться из так называемого *командного файла,* содержащего некоторую последовательность команд. Программный модуль ОС, ответственный за чтение отдельных команд или же последовательности команд из командного файла, иногда называют *командным интерпретатором*. Для компьютеров платформы IBM PC интерфейс командной строки обеспечивается семейством операционных систем MS-DOS.

Графические ОС реализует более сложный тип интерфейса, в котором в качестве органа управления кроме клавиатуры может использоваться мышь. Работа с графической операционной системой основана на взаимодействия активных и пассивных экранных элементов управления. В качестве активного элемента управления выступает *указатель мыши* – графический объект, перемещение которого на экране синхронизировано с перемещением мыши. В качестве пассивных элементов управления выступают графические *элементы управления приложений* (экранные кнопки, значки, переключатели, флажки, раскрывающиеся списки, строки меню и др.) Характер взаимодействия между активными и пассивными

элементами управления выбирает сам пользователь. В его распоряжении приемы наведения указателя мыши на элемент управления, щелчки кнопками мыши и другие средства.

Мы рассмотрели подсистемы управления ресурсами и подсистемы, общие для всех ресурсов.

1.3. Прочие функции операционной системы

1. Предоставление основных средств обслуживания компьютера – включение в базовый состав ОС первоочередных служебных приложений:

а) средства проверки дисков: средства логической проверки (то есть проверка целостности файловой структуры) и средства физической диагностики поверхности;

б) средства сжатия дисков;

в) средства кэширования дисков (кэш-память – сверхоперативная память). Так как взаимодействие процессора с дисками компьютера проходит намного медленнее операций обмена с оперативной памятью, ОС принимает специальные меры по сохранению части прочитанных с диска данных в оперативной памяти. В том случае, если по ходу работу процессору вновь потребуется обратиться к ранее считанным данным или программному коду, он может найти их в специальной области ОЗУ, называемой *дисковым кэшем*;

г) средства управления виртуальной памятью. Виртуальная память реализуется в виде файла подкачки (см. «ОС как виртуальная машина»);

д) средства резервного копирования данных (см. «Защита данных и администрирование»).

2. Возможность поддерживать функционирование локальной компьютерной сети без специального программного обеспечения.

3. Обеспечение доступа к основным службам Интернета средствами, интегрированными в состав ОС.

4. Возможность оформления рабочей среды ОС, в том числе и средствами, относящимся к категории мультимедиа.

5. Возможность обеспечения комфортной поочередной работы различных пользователей на одном персональном компьютере с сохранением персональных настроек рабочей среды каждого из них.

6. Возможность автоматического исполнения операций обслуживания компьютера и ОС по заданному расписанию или под управлением удаленного сервера.

На этом возможности ОС не исчерпываются. По мере развития аппаратных средств вычислительной техники и средств связи функции ОС непрерывно расширяются, а средства их исполнения совершенствуются.

1.4. Основные принципы построения ОС

1. *Частотный принцип*

Основан на выделении в алгоритмах программ и в обрабатываемых массивах действий и данных по частоте использования.

2. *Принцип модульности*

Под модулем понимают функциональный элемент рассматриваемой системы. Модуль предполагает легкий способ его замены на другой. Упорядочение модулей ОС значительно упрощает эксплуатацию и разработку, уменьшая число проектных ошибок

3. *Принцип функциональной избирательности*

Это логическое продолжение первых двух принципов. В ОС выделяется некоторая часть важных модулей, которые необходимы для эффективной организации вычислительного процесса – ядро ОС. Они используются наиболее часто.

4. *Принцип генерируемости*

Исходное представление системной программы ОС должно быть таковым, чтобы было возможно настраивать эту системную программу, исходя из конкретной конфигурации конкретной машины и круга решаемых проблем.

Генерация операционной системы – процесс создания конкретной операционной системы, учитывающий особенности конфигурации ЭВМ и задач пользователя.

5. *Принцип функциональной избыточности*

Этот принцип учитывает возможность проведения одной и той же работы различными средствами.

6. *Принцип «по умалчиванию»*

Основан на хранении в системе некоторых базовых описаний структур процессов, модулей, конфигурации оборудования, данных и т.д. Эту информацию ОС использует в качестве заданной, если пользователь не конкретизирует ее. При необходимости к «умалчиваемым» параметрам могут быть добавлены новые или изменены имеющиеся.

7. *Принцип перемещаемости*

Этот принцип предусматривает построение модулей, исполнение которых не зависит от места расположения в оперативной памяти

8. *Принцип защиты*

Определяет необходимость разработки мер, ограждающих программы и данные пользователей от искажений или нежелательных влияний друг на друга, а также пользователей на ОС и наоборот (например, способ защиты файлов – введение пароля).

9. *Принцип независимости программ от внешних устройств*

Принцип заключается в том, что связь программ с конкретными устройствами производится не на уровне трансляции программ, а в период планирования ее исполнения.

10. *Принцип открытой и наращиваемой ОС*

Открытая операционная система доступна для анализа пользователям и специалистам, обслуживающим вычислительную машину. Наращиваемая (развиваемая) ОС позволяет не только использовать возможности генерации, но и вводить в ее состав новые модули, совершенствовать существующие и т.д.

Контрольные вопросы

1. Поясните определение операционной системы как расширенной машины.
2. В чем состоит отличие в виртуальных машинах, предоставляемых операционной системой простому пользователю и прикладному программисту?
3. Сравните интерфейс прикладного программиста с операционной системой и интерфейс системного программиста с реальной аппаратурой. Что можно сказать о разнообразии и мощности интерфейсных функций, имеющихся в распоряжении каждого из них?
4. Расскажите о видах интерфейса пользователя, применяемых в разных операционных системах.
5. Опишите организацию хранения файлов на дисках компьютера.
6. В чем заключается операция установки приложения? Самостоятельно проведите установку какого-либо приложения.
7. Назовите абстрактно сформулированные задачи ОС по управлению любым типом ресурса. Конкретизируйте эти задачи применительно к процессору, памяти, внешним устройствам.

2. ЭВОЛЮЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ

(учебная проблема)

Контрольные вопросы

1. Какие события в развитии технической базы вычислительных машин стали вехами в истории операционных систем?
2. Что стало прообразом современных ОС?
3. В чем состояло принципиальное отличие первых мониторов пакетной обработки от уже существовавших к этому времени системных обрабатывающих программ — трансляторов, загрузчиков, компоновщиков, библиотек процедур?

4. В чем заключались недостатки ранних систем пакетной обработки?

5. Может ли компьютер работать без операционной системы?

6. Как эволюционировало отношение к концепции мультипрограммирования на протяжении всей истории ОС?

7. В каких вариантах было организовано мультипрограммирование?

8. Каковы преимущества мультипрограммных систем перед системами пакетной обработки?

9. В соответствии с определением ОС ее главными функциями являются предоставление удобств пользователю и эффективное управление ресурсами компьютера. Какая их этих двух функций должна была доминировать в мультипрограммных ОС времен IBM/360? А в первых ОС для персональных компьютеров?

10. Какова особенность многотерминального режима?

11. Каковы отличия ОС миникомпьютеров от ОС мэйнфреймов?

12. Какое влияние на развитие ОС оказал Интернет?

13. Чем объясняется особое место UNIX в истории операционных систем?

14. Что послужило стимулом для создания локальных сетей?

15. Когда появились первые персональные компьютеры?

16. Чем персональный компьютер отличается от миникомпьютера?

17. Как называлась первая многозадачная ОС для персональных компьютеров?

18. Какую роль в развитии ОС сыграла операционная система Windows?

19. В чем заключается принцип многопользовательского режима?

20. В чем состоит особенность ОС NetWare?

21. Каковы преимущества системы OS/2?

22. Что изменилось в конструкции и назначении ОС в период 90-х годов?

23. Опишите историю сетевых ОС.

24. Что такое корпоративная ОС?

25. Какую роль в развитии ОС играют справочные службы?

26. Что означает принцип многоплатформенности современных операционных систем?

27. Какую роль играет графический интерфейс в современной ОС?

28. В чем состоят современные тенденции развития ОС?

29. Какие ОС в настоящее время являются наиболее востребованными?

30. В чем причина масштабного завоевания рынка ОС Windows?

3. АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

Функциональная сложность операционной системы неизбежно приводит к сложности её архитектуры, под которой понимают структурную организацию ОС на основе различных программных модулей. Какой-либо единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС.

3.1. Модульная организация ОС

Наиболее общим подходом к структуризации ОС является разделение всех её модулей на две группы:

1. Ядро – модули, выполняющие основные функции ОС.
2. Модули, выполняющие вспомогательные функции ОС.

Модули ядра выполняют такие базовые функции ОС, как управление процессами, памятью, устройствами ввода-вывода и т.п. Ядро составляет сердцевину ОС, без него ОС является полностью не работоспособной и не сможет выполнить ни одну из своих функций.

В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса, такие как обработка прерываний, загрузка-выгрузка страниц и т.д. Эти функции недоступны для приложений. Другой класс функций ядра служит для поддержки приложений, создавая для них так называемую *прикладную программную среду*. Приложения могут обращаться к ядру с запросами – системными вызовами – для выполнения тех или иных действий, например для открытия и чтения файла, вывода графической информации на дисплей и т.д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования – API (см. «Интерфейс прикладного программирования»).

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями ОС, поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находится в оперативной памяти, то есть являются *резидентными.*

Обычно ядро оформляется в виде программного модуля некоторого специального формата, отличающегося от формата пользовательских приложений.

Часто бывает очень сложно провести четкую грань между операционной системой и пользовательскими приложениями, так как некоторые компоненты ОС оформлены как обычные приложения, то есть в виде исполняемых вспомогательных модулей стандартного для данной операционной системы формата. Решение о том, является ли какая-нибудь

программа частью ОС или нет, принимает производитель ОС. Некоторая программа может существовать определенное время как пользовательское приложение, а потом стать частью ОС, или наоборот.

Вспомогательные модули ОС обычно подразделяются на следующие группы:

1. Утилиты – программы, решающие отдельные задачи управления и сопровождения компьютерной системы (например, программы сжатия дисков и т.п.).

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

3. Программы предоставления пользователю дополнительных услуг – специальный вариант пользовательского интерфейса, калькулятор и даже игры.

4. Библиотеки процедур различного назначения, упрощающие разработку приложений (например, библиотека математических функций, функций ввода-вывода и т.п.).

Как и обычные приложения, для выполнения своих задач утилиты, обрабатывающие программы и библиотеки ОС, обращаются к функциям ядра посредством системных вызовов (рис.1).

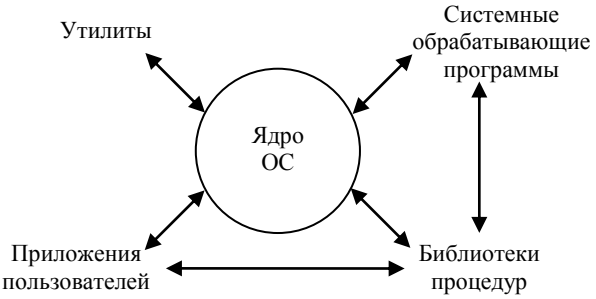

Рис.1. Ядро и вспомогательные модули ОС

Разделение операционной системы на ядро и модули-приложения обеспечивает легкую расширяемость ОС. Чтобы добавить новую высокоуровневую функцию, достаточно разработать новое приложение, и при этом не требуется модифицировать ответственные функции, образующие ядро системы. Однако внесение изменений в функции ядра может оказаться гораздо сложнее, и сложность эта зависит от структурной организации самого ядра. В некоторых случаях каждое исправление ядра может потребовать его полной перекомпиляции.

Модули ОС, оформленные в виде утилит, системных обрабатывающих программ и библиотек, обычно загружаются в оперативную па-

мять только на время выполнения своих функций, то есть являются *транзитными.* Постоянно в оперативной памяти располагаются только самые необходимые коды ОС, составляющие её ядро. Такая организация ОС экономит оперативную память компьютера.

Важным свойством архитектуры ОС, является возможность защиты кодов и данных ОС за счет выполнения функций ядра в привилегированном режиме.

3.2. Привилегированный режим ядра ОС

Для надежного управления ходом выполнения приложений ОС должна иметь по отношению к приложениям определенные привилегии. Иначе не корректно работающее приложение может вмешаться в работу ОС и, например, разрушить часть её кодов. Операционная система должна обладать исключительными полномочиями также для того, чтобы играть роль арбитра в споре приложений за ресурсы компьютера в мультипрограммном режиме. Ни одно приложение не должно без ведома ОС получать дополнительную область памяти, занимать процессор дольше разрешенного операционной системой периода времени, непосредственно управлять совместно используемыми внешними устройствами.

Обеспечить привилегии операционной системе невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум два режима работы – *пользовательский режим* и *привилегированный режим*, который также называют *режимом ядра*, или *режимом супервизора*. Подразумевается, что ОС или некоторые её части работают в привилегированном режиме, а приложения в пользовательском режиме.

Так как ядро выполняет все основные функции операционной системы, то чаще всего именно ядро становится той частью ОС, которая работает в привилегированном режиме. Приложения ставятся в подчиненное положение за счет запрета выполнения в пользовательском режиме некоторых критичных команд, связанных с переключением процессора с задачи на задачу, управлением устройств ввода-вывода, доступом к механизмам распределения и защиты памяти. Выполнение некоторых инструкций в пользовательском режиме запрещается безусловно (например, инструкция перехода в привилегированный режим), тогда как другие запрещается выполнять только при определенных условиях, и эти условия находятся под полным контролем ОС.

Между количеством уровней привилегий реализуемых аппаратно, и количеством уровней привилегий поддерживаемых ОС, нет прямого соответствия. Так, на базе четырех уровней, обеспечиваемых процессорами компании Intel, операционная система OS/2 строит трёхуровневую

систему привилегий, а ОС Windows NT, UNIX и некоторые другие ограничиваются двухуровневой системой.

С другой стороны, если аппаратура поддерживает хотя бы два уровня привилегий, то ОС может на этой основе создавать программным способом сколь угодно развитую систему защиты. Эта система может поддерживать несколько уровней привилегий, образующих иерархию. Примером сложной системы индивидуальной защиты ресурсов является типичная система защиты файлов и каталогов. Такая система позволяет задать для любого пользователя определённые права доступа к каждому из файлов и каталогов.

Повышение устойчивости ОС, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления выполнения системных вызовов. При системном вызове привилегированного ядра процессор должен переключиться из пользовательского режима в привилегированный, а при возврате к приложению – из привилегированного в пользовательский. Во всех типах процессоров из-за дополнительной двукратной задержки переключения переход на процедуру со сменой режима выполняется медленнее, чем вызов процедуры без смены режима.

Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической. Её используют многие популярные ОС, в том числе многочисленные версии UNIX, IBM OS/390, OS/2, и с определенными модификациями – Windows NT.

В некоторых случаях разработчики ОС отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, известная специализированная операционная система Net Ware компании Novell использует привилегированный режим процессоров Intel ×86 /Pentium как для работы ядра, так и для работы своих специфических приложений – загружаемых модулей NLM. При таком построении ОС обращения приложений к ядру выполняются быстрее, так как нет переключения режимов, но при этом отсутствует надежная защита памяти, занимаемой модулями ОС, от некорректно работающего приложения. Разработчики Net Ware пошли на такое потенциальное снижение надежности своей ОС, поскольку ограниченный набор её специализированных приложений позволяет компенсировать этот архитектурный недостаток за счет тщательной отладки каждого приложения.

В одном режиме работают также ядро и приложения тех операционных систем, которые разработаны для процессоров, вообще не поддерживающих привилегированного режима работы. Наиболее популярным процессором такого типа был процессор Intel 8088/86, послуживший

основой для персональных компьютеров компании IBM. Операционная система MS-DOS, разработанная компанией Microsoft для этих компьютеров, состояла из двух модулей MSDOS.SYS и IO.SYS, составлявших ядро системы. К этим модулям с системными вызовами обращались командный интерпретатор COMMAND.COM, системные утилиты и приложения. Некорректно написанные приложения вполне могли разрушить основные модули MS-DOS, что иногда и происходило, но область использования MS-DOS и не предъявляла высоких требований к надежности ОС.

3.3. Многослойная структура ОС

Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев: нижний слой – аппаратура, промежуточный – ядро, верхний – утилиты, обрабатывающие программы и приложения. Слоистую структуру вычислительной системы принято изображать в виде системы концентрических окружностей (рис.2), иллюстрируя тот факт, что каждый слой может взаимодействовать только со смежными слоями (приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра).

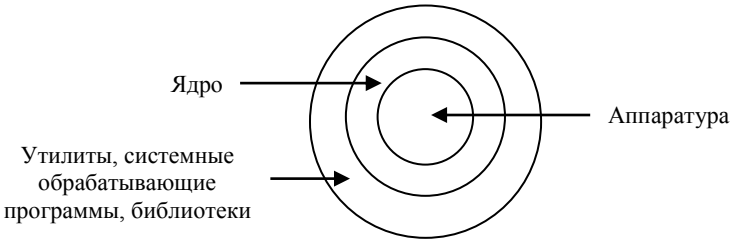

Рис.2. Структура вычислительной системы

Многослойный подход является универсальным и эффективным способом создания сложных систем любого типа, в том числе и программных. На основе функций нижележащего слоя следующий (вверх по иерархии) слой строит свои функции – более сложные и более мощные. Строгие правила касаются только взаимодействия между слоями системы, а между модулями внутри слоя связи могут быть произвольными. Отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться к другому модулю своего слоя, либо обратиться за помощью к нижележащему слою через межслойный интерфейс.

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

Поскольку ядро представляет собой сложный многофункциональный комплекс, то многослойный подход распространяется и на структуру ядра.

Ядро может состоять из следующих слоев:

1. *Средства аппаратной поддержки ОС*

До сих пор об ОС говорилось как о комплексе программ, но часть функций ОС может выполняться и аппаратными средствами. К операционной системе относят, естественно, не все аппаратные устройства компьютера, а только средства аппаратной поддержки ОС, то есть те, которые прямо участвуют в организации вычислительных процессов: средства поддержки привилегированного режима, систему прерываний, средства защиты областей памяти и т.п.

2. *Машинно-зависимые компоненты ОС*

Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС.

3. *Базовые механизмы ядра*

Этот слой выполняет наиболее примитивные операции ядра, такие как программное переключение контекстов процессов, диспетчеризацию прерываний, перемещение страниц из памяти на диск и обратно и т.п. Модули данного слоя не принимают решений о распределении ресурсов – они только отрабатывают принятые «наверху» решения, что и дает повод называть их исполнительными механизмами для модулей верхних слоев.

4. *Менеджеры ресурсов*

Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на данном слое работают менеджеры (называемые также диспетчерами) процессов, ввода-вывода, файловой системы и оперативной памяти. Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их

распределение в соответствии с запросами приложений. Например, менеджер виртуальной памяти управляет перемещением страниц из оперативной памяти на диск и обратно. Менеджер должен отслеживать интенсивность обращений к страницам, время пребывания их в памяти, и многие другие параметры, на основании которых он время от времени принимает решение о том, какие страницы необходимо выгрузить и какие – загрузить. Для исполнения принятых решений менеджер обращается к нижележащему слою базовых механизмов с запросом о загрузке (или выгрузке) конкретных страниц. Внутри слоя менеджеров существуют тесные взаимные связи, отражающие тот факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам – процессору, области памяти, к определенному файлу или устройству ввода-вывода.

5. *Интерфейс системных вызовов*

Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс операционной системы. Функции API, обслуживающие системные вызовы, представляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Для осуществления таких комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов, причем для выполнения одного системного вызова может понадобиться несколько таких обращений.

Если схематично изобразить структуру ядра ОС, то она будет выглядеть как на рис.3.

Рис.3. Многослойная структура ядра ОС

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними может быть и иным. Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению работы ядра за счет дополнительных накладных

расходов на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и логичность системы. Обычно операционные системы, прошедшие долгий путь эволюционного развития, например многие версии UNIX, имеют ядро с небольшим числом четко выделенных слоев, а у сравнительно «молодых» ОС, таких как Windows NT, ядро разделено на большее число слоев.

3.4. Аппаратная зависимость и переносимость ОС

Многие ОС успешно работают на различных аппаратных платформах без существенных изменений в своем составе. Это объясняется тем, что, несмотря на различие в деталях, средства аппаратной поддержки ОС большинства компьютеров приобрели сегодня много типовых черт, а именно эти средства в первую очередь влияют на работу компонентов операционной системы. В результате в ОС можно выделить достаточно компактный слой машинно-зависимых компонентов ядра и сделать остальные слои ОС общими для разных аппаратных платформ.

3.4.1. Типовые средства аппаратной поддержки ОС

Четкой границы между программной и аппаратной реализацией функций ОС не существует – решение о том, какие функции ОС будут выполняться программно, а какие аппаратно, принимается разработчиками аппаратного и программного обеспечения компьютера. Тем не менее, практически все аппаратные платформы имеют типичный набор средств аппаратной поддержки ОС:

1. *Средства поддержки привилегированного режима*

Основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора. Этот регистр содержит некоторые признаки, определяющие режимы работы процессора, в том числе и признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды.

Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро – пользователь) или четыре (например, у процессоров Intel ×86 /Pentium).

2. *Средства трансляции адресов*

Выполняют операции преобразования виртуальных адресов, которые содержатся в кодах процесса, в адреса физической памяти. Таблицы, предназначенные для трансляции адресов, обычно имеют большой объем, поэтому для их хранения используются области оперативной памяти, а аппаратура процессора содержит только указатели на эти области.

Средства трансляции адресов используют данные указатели для доступа к элементам таблиц и аппаратного выполнения алгоритма преобразования адреса, что значительно ускоряет процедуру трансляции по сравнению с ее чисто программной реализацией.

3. *Средства переключения процессов*

Предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным (контекст – информация о состоянии операционной среды процесса: см. лекцию «Управление процессами»). Для хранения контекстов приостановленных процессов обычно используются области оперативной памяти, которые поддерживаются указателями процессора. Переключение контекста выполняется по определенным командам процессора, например по команде перехода на новую задачу. Такая команда вызывает автоматическую загрузку данных из сохраненного контекста в регистры процессора, после чего процесс продолжается с прерванного ранее места.

4. *Система прерываний*

Позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, быстро переходить с одной программы на другую. Механизм прерываний нужен для того, чтобы оповестить процессор о возникновении в вычислительной системе некоторого непредсказуемого события или события, которое не синхронизировано с циклом работы процессора (например, некорректное завершение арифметической операции – переполнение регистра; завершение операции ввода-вывода внешним устройством – запись данных контроллером диска). При возникновении условий прерывания его источник (арифметический блок процессора, контроллер внешнего устройства и т.п.) выставляет определенный электрический сигнал. Этот сигнал прерывает выполнение процессором последовательности команд и вызывает автоматический переход на заранее определенную процедуру, называемую *процедурой обработки прерываний*. После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода (см. п.5.4.«Прерывания»).

Прерывания играют важную роль в работе любой ОС, являясь ее движущей силой. Действительно, большая часть действий ОС представлена прерываниями различного типа. Даже системные вызовы от приложений выполняются на многих аппаратных платформах с помощью специальной инструкции прерывания, вызывающей переход к выполнению соответствующих процедур ядра.

5. *Системный таймер*

Часто реализуется в виде быстродействующего регистра-счетчика, необходим ОС для выдержки интервалов времени. Системный таймер используется операционной системой в первую очередь для слежения за тем, как отдельные процессы расходую время процессора. Частота таймера, как правило, тесно связана с частотой тактового генератора процессора. Но не следует путать таймер ни с тактовым генератором, который вырабатывает сигналы, синхронизирующие все операции в компьютере, ни с системными часами – работающей на батареях электронной схеме, которые ведут независимый отсчет времени и календарной даты.

6. *Средства защиты областей памяти*

Обеспечивают на аппаратном уровне проверку возможности программного модуля осуществлять с данными определенной области памяти такие операции, как чтение, запись или выполнение. Если аппаратура компьютера поддерживает механизм трансляции адресов, то средства защиты областей памяти встраиваются в этот механизм.

3.4.2. Машинно-зависимые компоненты ОС

Одна и та же ОС не может без каких-либо изменений устанавливаться на компьютерах, отличающихся типом процессора или способом организации всей аппаратуры. В модулях ядра ОС не могут не отразиться такие особенности аппаратной платформы, как количество типов прерываний, особенности подключения внешних устройств и многие другие.

Однако опыт разработки ОС показывает: ядро можно спроектировать таким образом, что часть модулей будут машинно-зависимыми, а остальные не будут зависеть от особенностей аппаратной платформы. В хорошо структурированном ядре машинно-зависимые модули локализованы и образуют программный слой, примыкающий к слою аппаратуры. Такая локализация машинно-зависимых модулей существенно упрощает перенос ОС на другую аппаратную платформу.

Объем машинно-зависимых компонентов операционной системы зависит от того, насколько велики отличия в аппаратных платформах, для которых разрабатывается ОС. Например, ОС, построенная на 32-битовых адресах, для переноса на машину с 16-битовыми адресами должна быть практически переписана заново. Одно из наиболее очевидных отличий – несовпадение системы команд процессоров – преодолевается достаточно просто. ОС программируется на языке высокого уровня, а затем соответствующим компилятором вырабатывается код для конкретного типа процессора. Однако во многих случаях различия в организации аппаратуры лежат гораздо глубже и преодолеть их таким образом не удается. Например, однопроцессорный и двухпроцессорный компьютеры требуют при-

менения в ОС совершенно разных алгоритмов распределения процессорного времени. Аналогично отсутствие аппаратной поддержки виртуальной памяти приводит к принципиальному отличию реализации подсистемы управления памятью. В таких случаях не обойтись без внесения в код операционной системы специфики аппаратной платформы, для которой эта ОС предназначается.

Для уменьшения количества машинно-зависимых модулей производители ОС обычно ограничивают универсальность машинно-независимых модулей. Это означает, что их независимость носит условный характер и распространяется только на несколько типов процессоров и созданных на основе этих процессоров аппаратных платформ. По такому пути пошли, например, разработчики ОС Windows NT, ограничив количество типов процессоров для своей системы четырьмя и поставляя различные варианты кодов ядра для однопроцессорных и многопроцессорных компьютеров.

Для компьютеров на основе процессоров Intel ×86 Pentium разработка экранирующего машинно-зависимого слоя ОС несколько упрощает за счет встроенной в постоянную память компьютера базовой системы ввода-вывода – BIOS. BIOS содержит драйверы для всех устройств, входящих в базовую конфигурацию компьютера: жестких и гибких дисков, клавиатуры, дисплея и т.д. Эти драйверы выполняют весьма примитивные операции с управляемыми устройствами, например чтение группы секторов данных с определенной дорожки диска, но за счет этих операций экранируются различия аппаратных платформ персональных компьютеров и серверов на процессорах Intel разных производителей. Разработчики ОС могут пользоваться слоем драйверов BIOS как частью машинно-зависимого слоя ОС, а могут и заменить все или часть драйверов BIOS компонентами ОС.

3.4.3. Переносимость операционной системы

Операционная система называется *переносимой*, или *мобильной*, если её код может быть сравнительно легко перенесен с процессора одного типа на процессор другого типа, а с аппаратной платформы одного типа на аппаратную платформу другого типа.

Для того чтобы обеспечить свойство мобильности ОС, разработчики должны следовать следующим правилам:

1. Большая часть кода должна быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить систему. Такими языками являются стандартизованные языки высокого уровня. Большинство переносимых ОС написано на языке С, который имеет много особенностей, полезных для разработки кодов операцион-

ной системы, и компиляторы которого широко доступны. Программа, написанная на ассемблере, является переносимой только в тех случаях, когда перенос ОС планируется на компьютер, обладающий той же системой команд. В остальных случаях ассемблер используется только для тех непереносимых частей системы, которые должны непосредственно взаимодействовать с аппаратурой, или для частей, которые требуют максимальной скорости (например, целочисленная арифметика повышенной точности).

2. Объем машинно-зависимых частей кода, которые непосредственно взаимодействуют с аппаратными средствами, должен быть по возможности минимизирован. Так, например, следует всячески избегать прямого манипулирования регистрами и другими аппаратными средствами процессора. Для уменьшения аппаратной зависимости разработчики ОС должны также исключить возможность использования по умолчанию стандартных конфигураций аппаратуры или их характеристик. Для осуществления всех необходимых действий по управлению аппаратурой должен быть написан набор аппаратно-зависимых функций. Каждый раз, когда какому-либо модулю ОС требуется выполнить некоторое действие, связанное с аппаратурой, он манипулирует абстрактными данными, используя соответствующую функцию из имеющегося набора. Когда ОС переносится, то изменяются только эти данные и функции, которые ими манипулируют.

3. Аппаратно-зависимый код должен быть надежно изолирован в нескольких модулях, а не быть распределенным по всей системе. Изоляции подлежат все части ОС, которые отражают специфику как процессора, так и аппаратной платформы в целом. Низкоуровневые компоненты ОС, имеющие доступ к процессорно-зависимым структурам данных и регистрам, должны быть оформлены в виде компактных модулей, которые могут быть заменены аналогичными модулями для других процессоров. Для снятия платформенной зависимости, возникающей из-за различий между компьютерами разных производителей, построенными на одном и том же процессоре, должен быть введен хорошо локализованный программный слой машинно-зависимых функций.

В идеале слой машинно-зависимых компонентов ядра полностью экранирует остальную часть ОС от конкретных деталей аппаратной платформы. В результате происходит подмена реальной аппаратуры некой виртуальной машиной, одинаковой для всех вариантов аппаратной платформы. Все слои ОС, которые лежат выше слоя машинно-зависимых компонентов, могут быть написаны для управления именно этой виртуальной аппаратурой. Таким образом, у разработчиков появляется возможность создавать один вариант машинно-независимой части ОС для всего набора поддерживаемых платформ.

3.5. Микроядерная архитектура

Концепция микроядерной архитектуры является альтернативой классическому способу построения операционной системы, рассмотренному выше, в соответствии с которым все основные функции операционной системы, составляющие многослойное ядро, выполняются в привилегированном режиме. Каждое приложение пользовательского режима работает в собственном адресном пространстве и защищено тем самым от какого-либо вмешательства других приложений. Код ядра, выполняемый в привилегированном режиме, имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций.

Суть микроядерной архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая *микроядром.* Микроядро защищено от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие базовые (но не все) функции ядра по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений и управлению устройствами ввода-вывода. Набор функций микроядра обычно соответствует функциям слоя базовых механизмов обычного ядра.

Все остальные более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме. Однозначного решения о том, какие из системных функций нужно оставить в привилегированном режиме, а какие перенести в пользовательский, не существует. В общем случае многие менеджеры ресурсов, являющиеся неотъемлемыми частями обычного ядра – файловая система, подсистемы управления виртуальной памятью и процессами, менеджер безопасности и т.п., становятся «периферийными» модулями, работающими в пользовательском режиме.

Работающие в пользовательском режиме менеджеры ресурсов имеют принципиальные отличия от традиционных утилит и обрабатывающих программ операционной системы. Утилиты и обрабатывающие программы вызываются в основном пользователями. Ситуации, когда одному приложению требуется выполнение функции или процедуры другого приложения, возникают крайне редко. Поэтому в операционных системах с классической архитектурой отсутствует механизм, с помощью которого одно приложение могло бы вызвать функции другого.

Совсем другая ситуация возникает, когда в форме приложения оформляется часть операционной системы. Основным назначением такого приложения является обслуживание запросов других приложений, например создание процесса, выделение памяти, проверка прав доступа к

ресурсу и т.д. Именно поэтому менеджеры ресурсов, вынесенные в пользовательский режим, называются *серверами* ОС, то есть модулями, основным назначением которых является обслуживание запросов локальных приложений и других модулей ОС. Очевидно, что для реализации микроядерной архитектуры необходимым условием является наличие в операционной системе удобного и эффективного способа вызова процедур одного процесса из другого. Поддержка такого механизма и является одной из главных задач микроядра.

Механизм обращения к функциям ОС, оформленным в виде серверов, выглядит следующим образом: клиент, которым может быть либо прикладная программа, либо другой компонент ОС, запрашивает выполнение некоторой функции у соответствующего сервера, посылая ему сообщение. Непосредственная передача сообщений между приложениями невозможна, так как их адресные пространства изолированы друг от друга. Микроядро, выполняющееся в привилегированном режиме, имеет доступ к адресным пространствам каждого из этих приложений и поэтому может работать в качестве посредника. Микроядро сначала передает сообщение, содержащее имя и параметры вызываемой процедуры нужному серверу, затем сервер выполняет запрошенную операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения. Таким образом, работа микроядерной операционной системы соответствует известной модели клиент-сервер, в которой роль транспортных средств выполняет микроядро.

Преимущества и недостатки микроядерной архитектуры

Операционные системы, основанные на концепции микроядра, обладают рядом достоинств и в высокой степени удовлетворяют большинству требований, предъявляемых к современным операционным системам.

1. *Переносимость* обусловлена тем, что весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.

2. *Расширяемость* микроядерной ОС. В традиционных операционных системах даже при наличии многослойной структуры нелегко удалить один слой и поменять его на другой по причине множественности и размытости интерфейсов между слоями. Добавление новых функций и изменение существующих требует хорошего знания операционной системы и больших затрат времени. В то же время ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС. Добавление новой подсистемы требует разработки нового приложения, что никак не затрагивает целостность микроядра. Микроядерная структура позволяет не только добавлять, но и сокращать число компонентов операционной системы, что также бывает

очень полезно. *Конфигурируемость* ОС при микроядерном подходе не вызывает никаких проблем и не требует особых мер – достаточно изменить файл с настройками начальной конфигурации системы или же остановить не нужные больше серверы в ходе работы обычными для остановки приложений средствами.

3. *Надежность* операционной системы. Каждый сервер выполнен в виде отдельного процесса в своей собственной области памяти и таким образом защищен от других серверов операционной системы в отличие от традиционной ОС, где все модули ядра могут влиять друг на друга. И если отдельный сервер терпит крах, то он может быть перезапущен без останова или повреждения остальных серверов ОС. Кроме того, поскольку серверы выполняются в пользовательском режиме, они не имеют непосредственного доступа к аппаратуре и не могут модифицировать память, в которой хранится и работает микроядро. Другим потенциальным источником повышения надежности ОС является уменьшенный объем кода микроядра по сравнению с традиционным ядром – это снижает вероятность появления ошибок программирования.

За эти достоинства приходится платить снижением производительности, и это является основным недостатком микроядерной архитектуры.

Производительность. При классической организации ОС (рис.4, а) выполнение системного вызова сопровождается двумя переключениями режимов, а при микроядерной организации (рис.4, б) - четырьмя. Таким образом, операционная система на основе микроядра при прочих равных условиях всегда будет менее производительной, чем ОС с классическим ядром. Именно по этой причине микроядерный подход не получил широкого распространения.

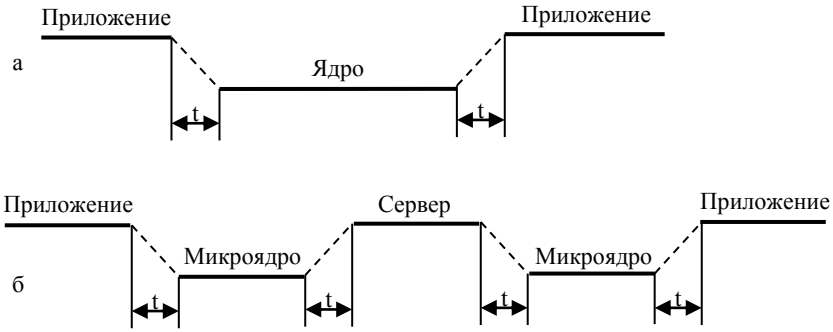

Рис.4. Смена режимов при выполнении системного вызова

Серьезность этого недостатка хорошо иллюстрирует история развития Windows NT. В версиях 3.1 и 3.5 диспетчер окон, графическая библиотека и высокоуровневые драйверы графических устройств входили в состав сервера пользовательского режима, и вызов функций этих модулей осуществлялся в соответствии с микроядерной схемой. Однако очень скоро разработчики Windows NT поняли, что такой механизм обращений к часто используемым функциям графического интерфейса существенно замедляет работу приложений и делает данную операционную систему уязвимой в условиях острой конкуренции. В результате в версию Windows NT 4.0 были внесены существенные изменения – все перечисленные выше модули были перенесены в ядро, что отдалило эту ОС от идеальной микроядерной архитектуры, но зато резко повысило ее производительность.

Таким образом, главная проблема, с которой сталкиваются разработчики операционной системы, решившие применить микроядерный подход, что включать в микроядро, а что выносить в пользовательское пространство. В идеальном случае микроядро может состоять только из средств передачи сообщений и средств взаимодействия с аппаратурой. Однако многие разработчики не всегда жестко придерживаются принципа минимизации функций ядра, часто жертвуя этим ради повышения производительности. В результате реализации ОС образуют некоторый спектр, на одном краю которого находятся системы с минимально возможным микроядром, а на другом – системы, подобные Windows NT, в которых микроядро выполняет достаточно большой объем функций.

3.6. Совместимость и множественные прикладные среды

В то время как многие архитектурные особенности операционных систем непосредственно касаются только системных программистов, концепция *множественных прикладных сред* непосредственно связана с нуждами конечных пользователей – возможностью операционной системы выполнять приложения, написанные для других операционных систем. Такое свойство операционной системы называется *совместимостью.*

Необходимо различать совместимость на двоичном уровне и совместимость на уровне исходных текстов. Приложения обычно хранятся в ОС в виде исполняемых файлов, содержащих двоичные образы кодов и данных. Двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее на выполнение в среде другой ОС.

Совместимость на уровне исходных текстов требует наличия соответствующего компилятора в составе программного обеспечения компьютера, на котором предполагается выполнять данное приложение, а также

совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция имеющихся исходных текстов в новый исполняемый модуль.

Совместимость на уровне исходных текстов важна в основном для разработчиков приложений, в распоряжении которых эти исходные тексты всегда имеются. Но для конечных пользователей практическое значение имеет только двоичная совместимость, так как только в этом случае они могут использовать один и тот же коммерческий продукт, поставляемый в виде двоичного исполняемого кода, в различных операционных средах и на различных машинах. Обладает ли новая ОС двоичной совместимостью или совместимостью исходных текстов с существующими операционными системами, зависит от многих факторов. Самый главный из них – архитектура процессора, на котором работает новая ОС. Если процессор использует тот же набор команд (возможно, с некоторыми добавлениями) и тот же диапазон адресов, тогда двоичная совместимость может быть достигнута довольно просто. Для этого достаточно соблюдения следующих условий:

– вызовы функций API, которые содержит приложение, должны поддерживаться данной операционной системой;

– внутренняя структура исполняемого файла приложения должна соответствовать структуре исполняемых файлов данной ОС.

Гораздо сложнее достичь двоичной совместимости операционным системам; предназначенным для выполнения на процессорах, имеющих разные архитектуры. Помимо соблюдения приведенных выше условий необходимо организовать *эмуляцию* двоичного кода. Для того чтобы компьютер смог интерпретировать машинные инструкции, которые ему изначально непонятны, на нем должно быть установлено специальное программное обеспечение – эмулятор.

Эмулятор должен последовательно выбирать каждую двоичную инструкцию одного процессора, программным способом дешифрировать ее, чтобы определить, какие действия она задает, а затем выполнять эквивалентную подпрограмму, написанную в инструкциях другого процессора. К тому же у второго процессора нет в точности таких же регистров, флагов и внутреннего арифметико-логического устройства, как у первого, поэтому он должен имитировать (эмулировать) и все эти элементы с использованием своих регистров или памяти. Это простая, но очень медленная работа.

Выходом в таких случаях является использование так называемых *прикладных программных сред.* Одной из составляющих, формирующих прикладную программную среду, является набор функций интерфейса прикладного программирования API, которые операционная система предоставляет своим приложениям. Для сокращения времени на

выполнение чужих программ прикладные среды имитируют обращения к библиотечным функциям.

Чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь обеспечить совместимость API. Концепции, положенные в основу разных ОС, могут входить в противоречие друг с другом. Например, в одной операционной системе приложению может быть разрешено непосредственно управлять устройствами ввода-вывода, в другой – эти действия являются прерогативой ОС. Каждая операционная система имеет свои собственные механизмы защиты ресурсов, свои алгоритмы обработки ошибок и исключительных ситуаций, особую структуру процесса и схему управления памятью, свою семантику доступа к файлам и графический пользовательский интерфейс. Для обеспечения совместимости необходимо организовать бесконфликтное сосуществование в рамках одной ОС нескольких способов управления ресурсами компьютера.

Способы реализации прикладных программных сред

Создание полноценной прикладной среды, полностью совместимой со средой другой операционной системы, является достаточно сложной задачей, тесно связанной со структурой операционной системы. Существуют различные варианты построения множественных прикладных сред, отличающиеся как особенностями архитектурных решений, так и функциональными возможностями, обеспечивающими различную степень переносимости приложений. Во многих версиях ОС UNIX транслятор прикладных сред реализуется в виде обычного приложения. В операционных системах, построенных с использованием микроядерной концепции, таких как, например, Windows NT, прикладные среды выполняются в виде серверов пользовательского режима. А в 0S/2 с ее более простой архитектурой средства организации прикладных сред встроены глубоко в операционную систему.

Варианты реализации множественных прикладных сред:

1. Основывается на стандартной многоуровневой структуре ОС. Операционная система поддерживает кроме своих «родных» приложений приложения других операционных систем. Для этого в ее составе имеются специальные приложения – прикладные программные среды, - которые транслируют интерфейсы «чужих» операционных систем в интерфейс своей «родной» операционной системы;

2. ОС имеет несколько равноправных прикладных программных интерфейсов. Для этого непосредственно в пространстве ядра системы размещены прикладные программные интерфейсы всех задействованных ОС. В этом варианте функции уровня API обращаются к функциям нижележащего уровня ОС, которые должны поддерживать все в общем случае несовместимые прикладные среды. В разных ОС по-разному осу-

ществляется управление системным временем, используется разный формат времени дня, на основании собственных алгоритмов разделяется процессорное время и т. д. Функции каждого API реализуются ядром с учетом специфики соответствующей ОС, даже если они имеют аналогичное назначение;

3. Основывается на микроядерном подходе. При этом очень важно отделить базовые, общие для всех прикладных сред, механизмы операционной системы от специфических для каждой из прикладных сред высокоуровневых функций, решающих стратегические задачи. В соответствии с микроядерной архитектурой все функции ОС реализуются микроядром и серверами пользовательского режима. Важно, что каждая прикладная среда оформляется в виде отдельного сервера пользовательского режима и не включает базовых механизмов. Приложения, используя API, обращаются с системными вызовами к соответствующей прикладной среде через микроядро. Прикладная среда обрабатывает запрос, выполняет его и отсылает приложению результат. Такому подходу к конструированию множественных прикладных сред присущи все достоинства и недостатки микроядерной архитектуры.

Контрольные вопросы

1. Какие из приведенных ниже терминов являются синонимами: привилегированный режим; защищенный режим; режим супервизора; пользовательский режим; реальный режим; режим ядра?

2. Можно ли, анализируя двоичный код программы, сделать вывод о невозможности ее выполнения в пользовательском режиме?

3. Каковы отличия в работе процессора в привилегированном и пользовательском режимах?

4. В идеале микроядерная архитектура ОС требует размещения в микроядре только тех компонентов ОС, которые не могут выполняться в пользовательском режиме. Что заставляет разработчиков операционных систем отходить от этого принципа и расширять ядро за счет перенесения в него функций, которые могли бы быть реализованы в виде процессов-серверов?

5. Какие этапы включает разработка варианта мобильной ОС для новой аппаратной платформы?

6. Опишите порядок взаимодействия приложений с операционной системой, имеющей микроядерную архитектуру.

7. Какими этапами отличается выполнение системного вызова в микроядерной ОС и ОС с монолитным ядром?

8. Может ли программа, эмулируемая на «чужом» процессоре, выполняться быстрее, чем на «родном»?

4. КЛАССИФИКАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ
(учебная проблема)

Контрольные вопросы

1. Почему, несмотря на повсеместное использование ОС, у пользователей зачастую возникают проблемы при попытке дать определение этому понятию?

2. Какие преимущества получили пользователи, получив возможность работать с операционными системами?

3. В чем состоят различия между операционными системами?

4. Чем обусловлены различия в организации и структуре ОС?

5. По каким признакам осуществляется классификация ОС?

6. Каковы особенности алгоритмов управления ресурсами компьютера, в соответствии с которыми различаются ОС?

7. Какие ОС вам известны в соответствии с особенностями использованного алгоритма управления процессором?

8. Что такое вытесняющая и невытесняющая многозадачность?

9. Чем отличается асимметричная ОС от симметричной?

10. Каковы характерные черты многонитевой ОС?

11. В чем состоят особенности систем разделения времени и систем реального времени?

12. Что такое кластер?

13. Как особенности методов построения ОС определяют их разнообразие?

14. Для чего нужно распределение организации ОС?

15. Какие возможности открываются благодаря наличию нескольких прикладных программных сред?

16. В каком случае процессор переходит от выполнения одной задачи к другой?

17. Почему в системах пакетной обработки невозможно гарантировать выполнение определенного задания в течение некоторого периода времени?

18. В каких ОС пропускная способность выше: в системах разделения времени или в системах пакетной обработки?

19. В чем состоит особенность мобильных ОС?

20. Каким образом выбор ОС связан с особенностями определенного компьютера?

21. Какие задачи ОС возникают при реализации сетевых функций?

22. Что является признаком эффективности ОС?

23. Как определить необходимую для конкретного пользователя ОС?

5. ПРОЦЕССЫ И ПОТОКИ

5.1. Мультипрограммирование

Способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ, носит название *мультипрограммирования, или многозадачности*. Эти программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, данные. Мультипрограммирование призвано повысить эффективность использования вычислительной системы, однако эффективность может пониматься поразному. Критериями эффективности вычислительных систем являются:

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

– *удобство работы пользователей,* заключающееся, в частности, в том, что они имеют возможность интерактивно работать одновременно с несколькими приложениями на одной машине.

– *реактивность системы* – способность системы выдерживать заранее заданные интервалы времени между запуском программы и получением результата.

В зависимости от выбранного критерия эффективности ОС делятся на системы пакетной обработки, системы разделения времени, и системы реального времени. Каждый тип ОС имеет специфические внутренние механизмы и особые области применения. Некоторые операционные системы могут поддерживать одновременно несколько режимов, например, часть задач может выполняться в режиме пакетной обработки, а часть – в режиме реального времени или в режиме разделения времени.

5.1.1. Мультипрограммирование в системах пакетной обработки

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность.

Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие разные требования к ресурсам, так, чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины. Например, в мультипро-

граммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в вычислительных системах, работающих под управлением пакетных ОС, невозможно гарантировать выполнение того или иного задания в течение определённого периода времени. Совмещение во времени вычислений и операций ввода-вывода может достигаться разными способами:

1) Способ характерен для компьютеров, имеющих специализированный процессор ввода-вывода. В компьютерах класса мэйнфреймов такие процессоры называют каналами. Обычно канал имеет систему команд, отличающуюся от системы команд центрального процессора. Эти команды специально предназначены для управления внешними устройствами, например, «установить начало листа», «напечатать строку». Канальные программы могут храниться в той же оперативной памяти, что и программы центрального процессора. В системе команд центрального процессора предусматривается специальная инструкция, с помощью которой каналу передаются параметры и указания на то, какую программу ввода-вывода он должен выполнить. Начиная с этого момента, центральный процессор и канал могут работать параллельно.

2) Способ реализуется в компьютерах, в которых внешние устройства управляются не процессором ввода-вывода, а контроллерами. Каждое внешнее устройство (или группа внешних устройств одного типа) имеет свой собственный контроллер, который автономно отрабатывает команды, поступающие от центрального процессора. При этом контроллер и центральный процессор работают асинхронно. Поскольку многие внешние устройства включают электромеханические узлы, контроллер выполняет свои команды управления устройствами существенно медленнее, чем центральный процессор – свои. Это обстоятельство используется для организации параллельного выполнения вычислений и операции ввода-вывода: в промежутке между передачей команд контроллеру центральный процессор может выполнять вычисления. Контроллер может сообщить центральному процессору о том, что он готов принять следующую команду, сигналом прерывания либо центральный процессор узнает об этом, периодически опрашивая состояние контроллера.

Максимальный эффект ускорения достигается при наиболее полном перекрытии вычислений и ввода-вывода.

Если в системе выполняются одновременно несколько задач, появляется возможность совмещения вычислений одной задачи с вводом-выводом другой. Пока одна задача ожидает какого-либо события (например, завершения ввода-вывода или загрузка с диска недостающей стра-

ницы программы), процессор не простаивает, как это происходит при последовательном выполнении программ, а выполняет другую задачу. Общее время выполнения смеси задач часто оказывается меньше, чем их суммарное время последовательного выполнения. Однако выполнение отдельной задачи в мультипрограммном режиме может занять больше времени, чем при монопольном выделении процессора этой задаче. При совместном использовании процессора в системе могут возникать ситуации, когда задача готова выполняться, но процессор занят выполнением другой задачи. В таких случаях задача, завершившая ввод-вывод, готова выполняться, но вынуждена ждать освобождения процессора, и это удлиняет срок её выполнения.

В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит по инициативе самой активной задачи, например, когда она отказывается от процессора из-за необходимости выполнить операцию ввода-вывода. Поэтому существует высокая вероятность того, что одна задача может надолго занять процессор и выполнение интерактивных задач станет невозможным. Взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводиться к тому, что он приносит задание, отдаёт его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат. Это повышает эффективность функционирования аппаратуры, но снижает эффективность работы пользователя.

5.1.2. Мультипрограммирование в системах разделения времени

Повышение удобства и эффективности работы пользователя является целью другого способа мультипрограммирования – разделение времени. В *системах разделения времени* пользователям (или одному пользователю) предоставляется возможность интерактивной работы сразу с несколькими приложениями. Для этого каждое приложение должно регулярно получать возможность «общения» с пользователем. Понятно, что в пакетных системах возможности диалога пользователя с приложением весьма ограничены. В системах разделения времени эта проблема решается за счёт того, что ОС принудительно периодически приостанавливает приложения, не дожидаясь, когда они добровольно освободят процессор. Всем приложениям попеременно выделяется квант процессорного времени, таким образом, пользователи, запустившие программы на выполнение, получают возможность работать с ними.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю

в этом случае предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго и время ответа оказывается приемлемым. Если квант выбран достаточно малым, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них является единственным. Системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая «выгодна» системе. Кроме того, производительность системы снижается из-за возросших накладных расходов вычислительной мощности на более частое переключение процессора с задачи на задачу. Это выполнение соответствует тому, что критериям эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя. Вместе с тем мультипрограммное выполнение интерактивных приложений повышает и пропускную способность компьютера. Аппаратура загружается лучше, поскольку в то время, пока одно приложение ждёт сообщения пользователя, другие приложения могут обрабатываться процессором.

5.1.3. Мультипрограммирование в системах реального времени

Системы реального времени, предназначены для управления с помощью компьютера различными техническими объектами (например, станком, спутником, научной экспериментальной установки и т.д.) или технологическими процессами (например, гальванической линией, доменным процессом и т.п.) Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная управляющая объектом программа. В противном случае может произойти авария: спутник выйдет из зоны видимости, экспериментальные данные, поступающие с датчиков, будут потеряны, толщина гальванического покрытия не будет соответствовать норме. Таким образом, критерием эффективности здесь является способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы – реактивностью. Требования ко времени реакции зависят от специфики управляемого процесса.

В системах реального времени используется ещё одна разновидность мультипрограммирования: здесь мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а

выбор программы на выполнение осуществляется по прерываниям (исходя из текущего состояния объекта) или в соответствии с расписанием плановых работ. Способность аппаратуры компьютера и ОС к быстрому ответу зависит в основном от скорости переключения с одной задачи на другую и, в частности, от скорости обработки сигналов прерывания. Если при возникновении прерывания процессор должен опросить сотни потенциальных источников прерывания, то реакция системы будет слишком медленной. Время обработки прерывания в системах реального времени часто определяет требования к классу процессора даже при небольшой его загрузке.

В системах реального времени не стремятся максимально загружать все устройства, наоборот, при проектировании программного управляющего комплекса обычно закладывается некоторый «запас» вычислительной мощности на случай пиковой нагрузки.

5.2. Мультипроцессорная обработка

Мультипроцессорная обработка – это способ организации вычислительного процесса в системах с несколькими процессорами, при котором несколько задач (процессов, потоков) могут одновременно выполняться на разных процессорах системы.

В настоящее время стало обычным включение нескольких процессоров в архитектуру даже персонального компьютера. Более того, многопроцессорность теперь являемся одним из необходимых требований, которые предъявляются к компьютерам, используемым в качестве центрального сервера более-менее крупной сети.

Не следует путать *мультипроцессорную* обработку с *мультипрограммной* обработкой. В мультипрограммных системах параллельная работа разных устройств позволяет одновременно вести обработку нескольких программ, но при этом в процессоре в каждый момент времени выполняется только *одна* программа. То есть в этом случае несколько задач выполняются попеременно на одном процессоре, создавая лишь видимость параллельного выполнения. А в мультипроцессорных системах *несколько задач* выполняются действительно *одновременно*, так как имеется несколько обрабатывающих устройств – процессоров. Конечно, мультипроцессирование вовсе не исключает мультипрограммирование: на каждом из процессоров может попеременно выполняться некоторый закреплённый заданным процессором набор задач.

Мультипроцессорная организация системы приводит к усложнению всех алгоритмов управления ресурсами, например, требуется планировать процессы не для одного, а для нескольких процессоров, что гораздо сложнее. Сложности заключаются и в возрастании числа конфликтов

по обращению к устройствам ввода-вывода, данным, общей памяти и совместно используемым программам. Необходимо предусмотреть эффективные средства блокировки при доступе к разделяемым информационным структурам ядра. Все эти проблемы должна решать операционная система путём синхронизации процессов, ведения очередей и планирования ресурсов. Более того, сама операционная система должна быть спроектирована так, чтобы уменьшить существующие взаимозависимости между собственными компонентами.

Мультипроцессорные системы часто характеризуются либо как симметричные, либо как несимметричные. При этом следует чётко определять, к какому аспекту мультипроцессорной системы относиться эта характеристика – к типу архитектуры или к способу организации вычислительного процесса.

Симметричная архитектура мультипроцессорной системы предполагает однородность всех процессов и единообразие включения процессов в общую схему мультипроцессорной системы. Традиционные симметричные мультипроцессорные конфигурации разделяют одну большую память между всеми процессорами. Масштабируемость, или возможность наращивания числа процессоров, в симметричных системах ограниченна, поскольку все они пользуются одной и той же оперативной памятью и, следовательно, должны располагаться в одном корпусе. Такая конструкция, называемая *масштабируемой по вертикали,* практически ограничивает число процессоров до четырёх или восьми.

В симметричных архитектурах все процессы пользуются одной и той же схемой отображения памяти. Они могут очень быстро обмениваться данными, так что обеспечивается достаточно высокая производительность для тех приложений (например, при работе с базами данных), в которых несколько задач должны активно взаимодействовать между собой.

В *асимметричной архитектуре* разные процессоры могут отличаться как своими характеристиками (производительностью, надёжностью, системой команд и т.д., вплоть до модели микропроцессора), так и функциональной ролью, которая поручается им в системе. Например, одни процессоры могут предназначаться для работы в качестве основных вычислителей, другие – для управления подсистемой ввода-вывода, третьи – ещё для каких-то особых целей. Функциональная неоднородность в ассиметричных архитектурах влечёт за собой структурные отличия во фрагментах системы, содержащих разные процессоры системы. Например, они могут отличаться схемами подключения процессоров к системной шине, набором периферийных устройств и способами взаимодействия процессоров с устройствами.

Масштабирование в асимметричной архитектуре реализуется иначе, чем в симметричной. Так как требование единого корпуса отсутствует, система может состоять из нескольких устройств, каждое из которых содержит один или несколько процессоров. Это *масштабирование по горизонтали*. Каждое такое устройство называется *кластером*, а вся мультипроцессорная система – кластерной.

Другим аспектом мультипроцессорных систем, который может характеризоваться симметрией или ее отсутствием, является способ организации вычислительного процесса, который, как известно, определяется и реализуется операционной системой.

Асимметричное мультипроцессирование является наиболее простым способом организации вычислительного процесса в системах с несколькими процессорами. Этот способ часто называют также «ведущий-ведомый». Функционирование системы по принципу «ведущий-ведомый» предполагает выделение одного из процессоров в качестве «ведущего», на котором работает операционная система и который управляет всеми остальными «ведомыми» процессорами. То есть, ведущий процессор берет на себя функции распределения задач и ресурсов, а ведомые процессоры работают только как обрабатывающие устройства и никаких действий по организации работы вычислительной системы не выполняют. Поскольку операционная система работает только на одном процессоре и функции управления полностью централизованы, то такая операционная система оказывается не намного сложнее ОС однопроцессорной системы.

Асимметричная организация вычислительного процесса может быть реализована как для симметричной мультипроцессорной архитектуры, в которой все процессоры аппаратно неразличимы, так и для несимметричной, для которой характерна неоднородность процессоров, их специализация на аппаратном уровне.

В архитектурно-асимметричных системах на роль ведущего процессора может быть назначен наиболее надёжный и производительный процессор. Если в наборе процессоров имеется специализированный процессор, ориентированный, например, на матричные вычисления, то при планировании процессов операционная система, реализующая асимметричное мультипроцессирование, должна учитывать специфику этого процессора. Такая специализация снижает надёжность системы в целом, так как процессоры не являются взаимозаменяемыми.

Симметричное мультипроцессирование как способ организации вычислительного процесса может быть реализовано в системах только с симметричной архитектурой. В таких системах процессоры работают с общими устройствами и разделяемой основной памятью. Симметричное мультипроцессирование реализуется общей для всех процессоров опера-

ционной системой. При симметричной организации все процессоры равноправно участвуют в управлении вычислительным процессором и в выполнении прикладных задач. Например, сигнал прерывания от принтера, который распечатывает данные прикладного процесса, выполняемого на некотором процессоре, может быть обработан совсем другим процессором. Разные процессоры могут в какой-то момент одновременно обслуживать как разные, так и одинаковые модули общей операционной системы. Для этого программы операционной системы должны обладать свойством повторной входимости (*реентабельностью).*

Операционная система полностью децентрализована. Модули ОС выполняются на любом доступном процессоре. Как только процессор завершает выполнение очередной задачи, он передаёт управление планировщику задач, который выбирает из общей для всех процессоров системной очереди задачу, которая будет выполняться на данном процессоре следующей. Все ресурсы выделяются для каждой выполняемой задачи по мере возникновения их потребностей и никак не закрепляются за процессором. При таком подходе все процессоры работают с одной и той же динамически выравниваемой нагрузкой. В решении одной задачи могут участвовать сразу несколько процессоров, если она допускает такое распараллеливание, например, путём представления в виде нескольких потоков. В случае отказа одного из процессоров симметричные системы, как правило, сравнительно просто реконфигурируются, что является их большим преимуществом перед плохо реконфигурируемыми асимметричными системами.

Симметричная и асимметричная организация вычислительного процесса в мультипроцессорной системе не связана напрямую с симметричной или асимметричной архитектурой, она определяется типом операционной системы. Так, в симметричных архитектурах вычислительный процесс может быть организован как симметричным образом, так и асимметричным. Однако асимметричная архитектура непременно влечёт за собой и асимметричный способ организации вычислений.

5.3. Планирование процессов и потоков

5.3.1. Создание процессов и потоков

Чтобы поддерживать мультипрограммирование, ОС должна определить и оформить для себя те внутренние единицы работы, между которыми будет разделяться процессор и другие ресурсы компьютера. В настоящее время в большинстве операционных систем определены два типа единиц работы. Более крупная единица работы, обычно носящая название *процесса*, или *задачи*, требует для своего выполнения несколь-

ких более мелких работ, для обозначения которых используют термины «поток», или «нить».

Любая работа вычислительной системы заключается в выполнении некоторой программы. Поэтому и с процессом, и с потоком связывается определённый программный код, который для этих целей оформляется в виде исполняемого модуля. В операционных системах, где существуют и процессы, и потоки, процесс рассматривается операционной системой как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени. Этот последний важнейший ресурс распределяется операционной системой между другими единицами работы – потоками, которые и получили своё название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд.

В простейшем случае процесс состоит из одного потока, и именно таким образом трактовалось понятие «процесс» до середины 80-х годов прошлого столетия (например, в ранних версиях UNIX) и в таком же виде оно сохранилось в некоторых современных ОС. В таких системах понятие «поток» полностью поглощается понятием «процесс», то есть остаётся только одна единица работы и потребления ресурсов – процесс. Мультипрограммирование осуществляется в таких ОС на уровне процессов.

Для того чтобы процессы не могли вмешаться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей задачей ОС является *изоляция* одного процесса от другого. Для этого операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса. При необходимости взаимодействия процессы обращаются к операционной системе, которая, выполняя функции посредника, предоставляет им средства межпроцессной связи – конвейеры, почтовые ящики, разделяемые секции памяти и некоторые другие.

В системах, в которых отсутствует понятие потока, возникают проблемы при организации параллельных вычислений в рамках процесса. А такая необходимость может возникать. Например, при работе с текстовым редактором желательно иметь возможность совмещать набор нового текста с такими продолжительными по времени операциями, как переформатирование значительной части текста, печать документа или его сохранение на локальном или удалённом диске. Ещё одним примером необходимости распараллеливания является сетевой сервер баз данных. В этом случае параллелизм желателен как для обслуживания различных запросов к базе данных, так и для более быстрого выполнения отдельного запроса за счёт одновременного просмотра различных записей базы.

В операционной системе существует механизм распараллеливания вычислений, который учитывает связи между отдельными ветвями вы-

числений одного и того же приложения – механизм *многопоточной обработки.* При этом вводится новая единица работы – *поток* выполнения, а понятие «процесс» в значительной степени меняет смысл. Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем процессов. В отличие от процессов, которые принадлежат разным приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в меньшей степени, чем процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к операционной системе, им достаточно использовать общую память – один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга. Поэтому, мультипрограммирование более эффективно на уровне потоков, а не процессов.

Задача, оформленная в виде нескольких потоков в рамках одного процесса, может быть выполнена быстрее за счёт параллельного выполнения её отдельных частей. Например, если электронная таблица была разработана с учётом возможностей многопоточной обработки, то пользователь может запросить пересчёт своего рабочего листа и одновременно продолжать заполнять таблицу.

Под созданием процесса понимают, прежде всего, создание *описателя процесса,* в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе, необходимые операционной системе для управления им. В число таких сведений могут входить, например, идентификатор процесса (приоритет и право доступа) и т.п. Примерами описателей процесса являются *дескриптор процесса* в UNIX, *объект-процесс (*object-process) в Windows NT. Создание описателя процесса знаменует собой появление в системе ещё одного претендента на вычислительные ресурсы. Начиная с этого момента при распределении ресурсов ОС должна принимать во внимание потребности нового процесса.

Создание процесса включает загрузку кодов и данных исполняемой программы данного процесса с диска в оперативную память. Для этого ОС должна обнаружить местоположение такой программы на диске, перераспределить оперативную память и выделить память исполняемой программе нового процесса. Затем необходимо считать программу в

выделенные для нее участки памяти и, возможно, изменить параметры программы в зависимости от размещения в памяти. В системах с виртуальной памятью в начальный момент может загружаться только часть кодов и данных процесса, с тем чтобы «подкачивать» остальные по мере необходимости. При выполнении всех этих действий подсистема управления процессами тесно взаимодействует с подсистемой управления памятью и файловой системой.

В многопоточной системе при создании процесса ОС создаёт для каждого процесса как минимум один поток выполнения. При создании потока так же, как и при создании процесса, операционная система генерирует специальную информационную структуру – *описатель потока*, который содержит идентификатор потока, данные о правах доступа и приоритете, о состоянии потока и другую информацию. В исходном состоянии поток находится в приостановленном состоянии. Момент выборки потока на выполнение осуществляется в соответствии с принятым в данной системе правилом предоставления процессорного времени и с учетом всех существующих в данный момент потоков и процессов. В случае если коды и данные процесса находятся в области подкачки, необходимым условием активизации потока (процесса) является также наличие места в оперативной памяти для загрузки его исполняемого модуля.

Во многих системах поток может обратиться к операционной системе с запросом на создание так называемых потоков-потомков, часто порождение потомков является основным механизмом создания процессов и потоков. В разных ОС по-разному строятся отношения между потоками-потомками и их родителями. В одних ОС выполнение родительского потока синхронизируется с его потомками, например, после завершения родительского потока ОС может снимать с выполнения всех его потомков. В других системах потоки-потомки могут выполняться асинхронно по отношению к родительскому потоку. Потомки, как правило, наследуют многие свойства родительских потоков.

5.3.2. Планирование и диспетчеризация потоков

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

ниматься во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращения к вводу-выводу и другие факторы. ОС планирует выполнение потоков независимо от того, принадлежат ли они одному или разным процессам. Так, например, после выполнения потока некоторого процесса ОС может выбрать для выполнения другой поток того же процесса или же назначить к выполнению поток другого процесса.

Планирование потоков включает в себя решение двух задач:

– определение момента времени для смены текущего активного потока;

– выбор для выполнения потока из очереди готовых потоков.

Существует множество различных алгоритмов планирования потоков, по-своему решающих каждую из приведённых выше задач. Алгоритмы планирования могут преследовать различные цели и обеспечивать разное качество мультипрограммирования. Например, в одном случае выбирается такой алгоритм планирования, при котором гарантируется, что ни один поток (процесс) не будет занимать процессор дольше определённого времени, в другом случае целью является максимально быстрое выполнение «коротких» задач, а в третьем случае – преимущественное право занять процессор получают потоки интерактивных приложений. Именно особенности реализации планирования потоков в наибольшей степени определяют специфику операционной системы, в частности, является ли она системой пакетной обработки, системой разделения времени или системой реального времени.

В большинстве операционных систем универсального назначения планирование осуществляется *динамически* (on-line), то есть решения принимаются во время работы системы на основе анализа текущей ситуации. ОС работает в условиях неопределённости – потоки и процессы появляются в случайные моменты времени и так же непредсказуемо завершаются. Динамические планировщики могут гибко приспосабливаться к изменяющейся ситуации и не используют никаких предположений о мультипрограммной смеси. При этом операционная система должна затрачивать значительные усилия.

Другой тип планирования – *статический* – может быть использован в специализированных системах, в которых весь набор одновременно выполняемых задач определён заранее, например, в системах реального времени. *Планировщик* называется *статическим* (или предварительным планировщиком), если он принимает решения о планировании не во время работы системы, а заранее (off-line).

Результатом работы статического планировщика является таблица, называемая расписанием, в котором указывается, какому потоку (процессу), когда и на какое время должен быть предоставлен процессор. Для

построения расписания планировщику нужны как можно более полные предварительные данные о характеристиках набора задач, например, о максимальном времени выполнения каждой задачи, ограничениях предшествования и т.д.

Диспетчеризация – реализация найденного в результате планирования (динамического или статического) решения: переключение процессора с одного потока на другой. Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем, чтобы впоследствии использовать эту информацию для последующего возобновления выполнения данного потока. Контекст отражает, во-первых, состояние аппаратуры компьютера в момент прерывания потока: значение счётчика команд, содержимое регистров общего назначения, режим работы процессора, флаги, маски прерываний другие параметры. Кроме того, контекст включает параметры операционной среды, а именно ссылки на открытые файлы, данные о незавершённых операциях ввода-вывода, коды ошибок выполняемых данным потоком системных вызовов и т.д. Диспетчеризация – это сохранение контекста текущего потока, который требуется сменить; загрузка контекста нового потока, выбранного в результате планирования; запуск нового потока на выполнение.

Так как операция переключения контекстов существенно влияет на производительность вычислительной системы, программные модули ОС выполняют диспетчеризацию потоков совместно с аппаратными средствами процессора. В различных ОС можно встретить компоненты, имеющие названия *планировщик (scheduler)* или *диспетчер (dispatcher).* Чаще всего то и другое названия используется для обозначения компонентов, которые занимаются планированием.

ОС выполняет планирование потоков, принимая во внимание их *состояние.* В мультипрограммной системе поток может находиться в одном из трех основных состояний:

– *выполнение* – активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

– *ожидание* – пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например завершения операции ввода-вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса);

– *готовность* – также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

В течение своей жизни каждый поток переходит из одного состояния в другое в соответствии с алгоритмом планирования потоков, принятым в данной операционной системе.

Только что созданный поток находится в состоянии готовности, он готов к выполнению и стоит в очереди к процессору. Когда в результате планирования подсистема управления потоками принимает решение об активизации данного потока, он переходит в состояние выполнения и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние ожидания какого-нибудь события, либо будет принудительно «вытеснен» из процессора. В последнем случае поток возвращается в состояние готовности. В это же состояние поток переходит из состояния ожидания, после того как ожидаемое событие произойдет.

В состоянии выполнения в однопроцессорной системе может находиться не более одного потока, а в каждом из состояний ожидания и готовности – несколько потоков. Эти потоки образуют очереди соответственно ожидающих и готовых потоков. Очереди потоков организуются путем объединения в списки описателей отдельных потоков. Таким образом, каждый описатель потока, содержит, по крайней мере, один указатель на другой описатель, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать потоки, переводить потоки из одного состояния в другое.

5.3.3. Алгоритмы планирования потоков

Алгоритмы планирования потоков можно разделить на два основных класса: вытесняющие и невытесняющие.

Невытесняющие алгоритмы основаны на том, что активному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению поток.

Вытесняющие алгоритмы – это такие способы планирования потоков, в которых решение о переключении процессора с выполнения одного потока на выполнение другого потока принимается операционной системой, а не активной задачей.

Основным различием между вытесняющими и невытесняющими алгоритмами является степень централизации механизма планирования потоков. При вытесняющем мультипрограммировании функции планирования потоков целиком сосредоточены в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться одновременно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активного потока, запоминает его контекст, выбирает из

очереди готовых потоков следующий, запускает новый поток на выполнение, загружая его контекст.

При невытесняющем мультипрограммировании механизм планирования распределен между операционной системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения очередного цикла своего выполнения и только затем передает управление ОС с помощью какого-либо системного вызова. ОС формирует очереди потоков и выбирает в соответствии с некоторым правилом (например, с учетом приоритетов) следующий поток на выполнение. Такой механизм создает проблемы, как для пользователей, так и для разработчиков приложений. Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, пользователь не может переключиться с этой задачи на другую задачу. Поэтому разработчики приложений для операционной среды с невытесняющей многозадачностью вынуждены, возлагая на себя часть функций планировщика, создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить «дружественное» отношение своей программы к другим выполняемым одновременно с ней программам. Для этого в программе должны быть предусмотрены частые передачи управления операционной системе. Крайним проявлением «не дружественности» приложения является его зависание, которое приводит к общему краху системы.

В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный планирующий механизм имеет возможность снять зависшую задачу с выполнения. Однако распределение функций планирования потоков между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент возвращения управления, то при этом исключаются нерациональные прерывания программ в «неудобные» для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждого цикла выполнения использует их монопольно и уверена, что на протяжении этого периода никто другой не изменит данные. Существенным преимуще-

ством невытесняющего планирования является более высокая скорость переключения с потока на поток.

Почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений реализованы вытесняющие алгоритмы планирования потоков (процессов).

Алгоритмы планирования, основанные на квантовании

В основе многих вытесняющих алгоритмов планирования лежит концепция *квантования.* В соответствии с этой концепцией каждому потоку поочередно для выполнения предоставляется ограниченный непрерывный период процессорного времени – квант.

Смена активного потока происходит, если:

– поток завершился и покинул систему;

– произошла ошибка;

– поток перешел в состояние ожидания;

– исчерпан квант процессорного времени, отведенный данному потоку.

Поток, который исчерпал свой квант, переводится в состояние готовности и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый поток из очереди готовых для выполнения.

Кванты, выделяемые потокам, могут быть одинаковыми для всех потоков или различными. Типичное значение кванта в системах разделения времени составляет десятки миллисекунд. Если квант короткий, то суммарное время, которое проводит поток в ожидании процессора, прямо пропорционально времени, требуемому для его выполнения. При этом не учитывается, что потоки могут использовать кванты не полностью, что часть времени они могут тратить на ввод-вывод, что количество потоков в системе может динамически меняться и т. д.

Чем больше квант, тем выше вероятность того, что потоки завершатся в результате первого же цикла выполнения, и тем менее явной становится зависимость времени ожидания потоков от их времени выполнения. При достаточно большом кванте алгоритм квантования вырождается в алгоритм последовательной обработки, присущий однопрограммным системам, при котором время ожидания задачи в очереди вообще никак не зависит от ее длительности.

Кванты, выделяемые одному потоку, могут быть фиксированной величины, а могут и изменяться в разные периоды жизни потока. Потоки получают для выполнения квант времени, но некоторые из них используют его не полностью, например, из-за необходимости выполнить ввод или вывод данных. В результате возникает ситуация, когда потоки с интенсивными обращениями к вводу-выводу используют только небольшую часть выделенного им процессорного времени. Алгоритм планиро-

вания может исправить это. В качестве компенсации за неиспользованные полностью кванты потоки получают привилегии при последующем обслуживании. Для этого планировщик создает две очереди готовых потоков. Одна очередь образована потоками, которые пришли в состояние готовности в результате исчерпания кванта времени, а другая очередь – потоками, у которых завершилась операция ввода-вывода. При выборе потока для выполнения прежде всего просматривается вторая очередь, и только если она пуста, квант выделяется потоку из первой очереди.

Многозадачные ОС теряют некоторое количество процессорного времени для выполнения вспомогательных работ во время переключения контекстов задач. При этом запоминаются и восстанавливаются регистры, флаги и указатели стека, а также проверяется статус задач для передачи управления. Затраты на эти вспомогательные действия не зависят от величины кванта времени, поэтому чем больше квант, тем меньше суммарные накладные расходы, связанные с переключением потоков.

Алгоритмы планирования, основанные на приоритетах

Другой важной концепцией, лежащей в основе многих вытесняющих алгоритмов планирования, является *приоритетное обслуживание*. Приоритетное обслуживание предполагает наличие у потоков некоторой изначально известной характеристики – приоритета, на основании которой определяется порядок их выполнения. *Приоритет* – это число, характеризующее степень привилегированности потока при использовании ресурсов вычислительной машины, в частности процессорного времени: чем выше приоритет, тем выше привилегии, тем меньше времени будет проводить поток в очередях.

Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что приоритет потока тем выше, чем больше число, обозначающее приоритет. В других системах, наоборот, чем меньше число, тем выше приоритет. В большинстве операционных систем, поддерживающих потоки, приоритет потока непосредственно связан с приоритетом процесса, в рамках которого выполняется данный поток. Приоритет процесса назначается операционной системой при его создании. Значение приоритета включается в описатель процесса и используется при назначении приоритета потокам этого процесса. При назначении приоритета вновь созданному процессу ОС учитывает, является этот процесс системным или прикладным, каков статус пользователя, запустившего процесс, было ли явное указание пользователя на присвоение процессу определенного уровня приоритета. Поток может быть инициирован не только по команде пользователя, но и в результате выполнения системного вызова другим потоком. В этом случае при назначении приоритета новому потоку ОС должна принимать во внимание значение параметров системного вызова.

Во многих ОС предусматривается возможность изменения приоритетов в течение жизни потока. Изменения приоритета могут происходить по инициативе самого потока, когда он обращается с соответствующим вызовом к операционной системе, или по инициативе пользователя, когда он выполняет соответствующую команду. Кроме того, ОС сама может изменять приоритеты потоков в зависимости от ситуации, складывающейся в системе. В последнем случае приоритеты называются *динамическими* в отличие от неизменяемых, *фиксированных*, приоритетов.

От того, какие приоритеты назначены потокам, существенно зависит эффективность работы всей вычислительной системы. В современных ОС возможности пользователей влиять на приоритеты процессов и потоков ограничены. При этом обычные пользователи, как правило, не имеют права повышать приоритеты своим потокам, это разрешено делать (в определенных пределах) только администраторам. В большинстве же случаев ОС присваивает приоритеты потокам по умолчанию.

Существуют две разновидности приоритетного планирования: обслуживание с относительными приоритетами и обслуживание с абсолютными приоритетами. В обоих случаях выбор потока на выполнение из очереди готовых осуществляется одинаково: выбирается поток, имеющий наивысший приоритет**.** Однако проблема определения момента смены активного потока решается по-разному. В системах с относительными приоритетами активный поток выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ожидания (или произойдет ошибка, или поток завершится). В системах с абсолютными приоритетами выполнение активного потока прерывается кроме указанных выше причин, еще при одном условии: если в очереди готовых потоков появился поток, приоритет которого выше приоритета активного потока**.** В этом случае прерванный поток переходит в состояние готовности.

В системах, в которых планирование осуществляется на основе относительных приоритетов, минимизируются затраты на переключения процессора с одной работы на другую. Но здесь могут возникать ситуации, когда одна задача занимает процессор долгое время. Для систем разделения времени и реального времени такое обслуживание не подходит: интерактивное приложение может ждать своей очереди часами, пока вычислительной задаче не потребуется ввод-вывод. Относительные приоритеты используются широко в системах пакетной обработки. В системах с абсолютными приоритетами время ожидания потока в очередях может быть сведено к минимуму, если ему назначить самый высокий приоритет. Такой поток будет вытеснять из процессора все остальные потоки (кроме потоков, имеющих такой же наивысший приоритет). Это делает планирование на основе абсолютных приоритетов подходящим для систем управления объектами, в которых важна быстрая реакция на событие.

Смешанные алгоритмы планирования, перепланировка

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

Для реализации алгоритма планирования ОС должна получать управление всегда, когда в системе происходит событие, требующее перераспределения процессорного времени. К таким событиям относятся:

1. Прерывание от таймера, сигнализирующее, что время, отведенное активной задаче на выполнение, закончилось. Планировщик переводит задачу в состояние готовности и выполняет перепланирование.

2. Активная задача выполнила системный вызов, связанный с запросом на ввод-вывод или на доступ к ресурсу, который в настоящий момент занят. Планировщик переводит задачу в состояние ожидания и выполняет перепланирование.

3. Активная задача выполнила системный вызов, связанный с освобождением ресурса. Планировщик проверяет, не ожидает ли этот ресурс какая-либо задача. Если да, то эта задача переводится из состояния ожидания в состояние готовности. При этом возможно, что задача, которая получила ресурс, имеет более высокий приоритет, чем текущая активная задача. После перепланирования более приоритетная задача получает доступ к процессору, вытесняя текущую задачу.

4. Внешнее (аппаратное) прерывание, которое сигнализирует о завершении периферийным устройством операции ввода-вывода, переводит соответствующую задачу в очередь готовых, и выполняется планирование.

5. Внутреннее прерывание сигнализирует об ошибке, которая произошла в результате выполнения активной задачи. Планировщик снимает задачу и выполняет перепланирование.

При возникновении каждого из этих событий планировщик выполняет просмотр очередей и решает вопрос о том, какая задача будет выполняться следующей. Помимо указанных существует и ряд других событий (часто связанных с системными вызовами), требующих перепланировки. Например, запросы приложений и пользователей на созда-

ние новой задачи или повышение приоритета уже существующей задачи создают новую ситуацию, которая требует пересмотра очередей и, возможно, переключения процессора.

5.4. Прерывания

5.4.1. Назначение и типы прерываний

Прерывание также является важным понятием в системном программировании, поскольку оно тесно связано с состоянием процесса и переключением процессов. Любая система управления процессами использует прерывания.

Прерывание – асинхронная прозрачная процедура (функция), вызываемая по внешнему событию. Прежде всего, прерывание является процедурой, то есть некоторым алгоритмом, после выполнения которого происходит возврат к той части основной программы (точнее к тому состоянию процесса), в котором она была вызвана. Понятие «асинхронная процедура» означает, что ее вызов осуществляется независимо от хода выполнения основной программы. Условием вызова ее является наступление некоторого внешнего по отношению к процессу события (например, поступление входных данных от внешнего устройства).

Прерывания, возникающие при работе вычислительной системы, можно разделить на несколько групп (рис.5).

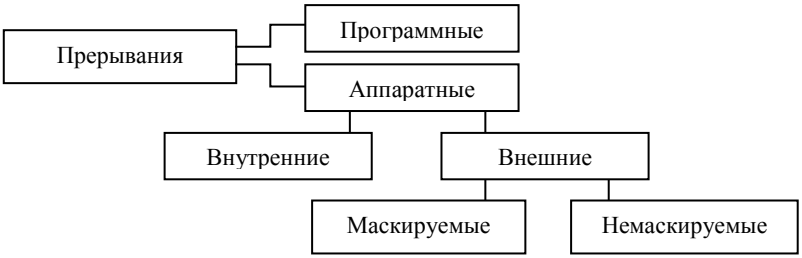

Рис. 5. Типы прерываний

Аппаратные прерывания вызываются физическими устройствами и в общем случае невозможно предсказать, когда и по какой причине программа будет прервана. Аппаратные прерывания не координируются с работой программного обеспечения.

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

– при нарушении адресации (например, в адресной части выполняемой команды указан запрещенный или несуществующий адрес или обращение к отсутствующему сегменту или странице при организации механизмов виртуальной памяти);

– при наличии в поле кода не задействованной двоичной комбинации;

– при делении на нуль;

– при переполнении или исчезновении порядка;

– при обнаружении средствами контроля ошибок четности, ошибок в работе различных устройств аппаратуры.

Внешние прерывания возникают по сигналу какого-либо внешнего устройства, например:

– прерывание, которое информирует систему о том, что требуемый сектор диска уже прочитан, его содержимое доступно программе;

– прерывание, которое информирует систему о том, что завершилась печать символа на принтере и необходимо выдать следующий символ;

– прерывания по нарушению питания и др.

Маскируемые и немаскируемые внешние прерывания

Часто при выполнении критических участков программ, для того чтобы гарантировать выполнение определенной последовательности команд целиком, приходится запрещать прерывания (т.е. сделать систему нечувствительной ко всем или отдельным прерываниям). Наличие сигнала прерывания не обязательно должно вызывать прерывание исполняющейся программы. Процессор может обладать средствами защиты от прерываний: отключение системы прерываний, маскирование (запрет) отдельных сигналов прерываний. Прерывания, которые замаскировать нельзя – это немаскируемые прерывания.

Аппаратные прерывания вырабатываются устройствами компьютера, когда возникает необходимость их обслуживания. В отличие от программных прерываний, вызываемых запланировано самой прикладной программой, аппаратные прерывания всегда происходят асинхронно по отношению к выполняющимся программам. Кроме того, может возникнуть одновременно несколько прерываний. Выбор одного из них для обработки осуществляется на основе приоритетов, приписанных каждому типу прерывания. Если происходит одновременно несколько прерываний, то система отдает предпочтение самому высокоприоритетному, откладывая на время обработку остальных прерываний.

Уровни приоритетов обозначаются сокращенно IRQ0 – IRQ15 или IRQ0 – IRQ23 (в зависимости от микросхемы реализации). Например, прерыванию времени суток дан максимальный приоритет, поскольку,

если оно будет постоянно теряться, то будут неверными показания системных часов.

Программные прерывания

Программы могут сами вызывать прерывания с заданным номером. Для этого они используют команду INT (в процессорах Intel Pentium). По этой команде процессор осуществляет практически те же действия, что и при обычных прерываниях, но только это происходит в предсказуемой точке программы – там, где программист поместил данную команду. Поэтому *программные прерывания* не являются асинхронными (программа «знает», когда она вызывает прерывание).

Программные прерывания в прямом смысле прерываниями не являются, поскольку представляют собой лишь специфический способ вызова процедур – не по адресу, а по номеру в таблице.

Механизм программных прерываний был специально введен для того, чтобы переключение на системные программные модули происходило не просто как переход в подпрограмму, а точно таким же образом, как и обычные прерывания. Этим обеспечивается автоматическое переключение процессора в привилегированный режим с возможностью исполнения любых команд. Использование программных прерываний приводит к более компактному коду программ по сравнению с использованием стандартных команд выполнения процедур.

Примерами программных прерываний являются: привилегированная команда в режиме пользователя; адрес вне диапазона; нарушение защиты памяти; арифметическое переполнение; выход за границу сегмента.

В случае о прерывании самой программы обработки прерывания говорят о *вложенном прерывании*.

5.4.2. Механизм прерываний

С каждым прерыванием связывают число, называемое номером типа прерывания или просто номером прерывания. Система умеет распознавать, какое прерывание (с каким номером) произошло, и запускает соответствующую этому номеру программу обработки прерывания. Таким образом, при поступлении сигнала на прерывание происходит принудительная передача управления от выполняемой программы к системе, а через нее – к обработчику прерываний.

Обработчик прерываний – программа обработки прерывания, являющаяся частью ОС, предназначенная для выполнения ответных действий на условие, вызвавшее прерывание.

Адреса программ, соответствующих различным прерываниям, собраны в таблицу, которая называется *таблицей векторов прерываний*.

Микропроцессор простым способом определяет местоположение программы обработки прерывания: путем использования этой таблицы.

Таблица векторов прерываний занимает первый килобайт оперативной памяти – адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов – FAR-адресов обработчиков прерываний. Эти элементы называются *векторами прерываний*. В первом слове элемента таблицы записано смещение, а во втором – адрес сегмента обработчика прерывания. Векторами являются просто полные адреса памяти программы (в сегментированной форме), которая должна быть активизирована в случае возникновения прерывания.

После выполнения в ответ на запрос на прерывание любого требуемого действия стандартная программа обработки прерываний выполняет команду загрузки состояния процессора. В результате управление передается прерванной программе. Происходит это следующим образом: команда загрузки состояния процессора вызывает загрузку сохраненного содержимого слова состояния, счетчика команд и других регистров из соответствующих слов области сохранения, начиная с адреса, указанного в команде. Это приводит к восстановлению содержимого регистров и состояния процессора, которые были в момент прерывания. Управление затем передается на команду, перед выполнением которой произошло прерывание.

Сохранение и восстановление состояния процессора и содержимого регистров называют *операцией контекстного переключения*.

У большинства машин имеется так называемое слово состояния, которое содержит часть информации, используемой при обработке прерываний. Одним из элементов этого слова (например, первый) является признак, определяющий, в каком режиме находится процессор: *пользовательском или супервизора.*

Обычные программы находятся в пользовательском режиме (признак равен нулю). Когда происходит прерывание, новое загружаемое содержимое слово состояния имеет признак, равный 1, что автоматически переводит процессор в режим супервизора. В этом режиме становится возможным использование привилегированных команд. Перед тем, как значение слова состояния будет сохранено, в другом его элементе (например, втором) будет установлено значение, указывающее на причину прерывания:

– при программном прерывании отражается тип вызвавшего его условия, например деление на нуль;

– при прерывании по вводу-выводу заносится номер канала, вызвавший прерывание.

В третьем элементе указывается, выполняет ли процессор команды или простаивает. В четвертом элементе содержится указатель, идентифи-

цирующий текущую выполняемую программу. В пятом элементе содержится маска прерываний, которая используется для контроля над разрешением прерываний (поле MASK).

Это поле используется, чтобы не допустить наступления прерываний определенного типа, пока первое из них не будет обработано. В MASK каждый бит соответствует некоторому классу прерываний. Если какой-то бит установлен в 1, то прерывания соответствующего класса разрешены, если в 0, то запрещены. В последнем случае говорят, что они *маскированы* или *маскируемые* (их также называют запрещенными или закрытыми). Однако маскированные прерывания не теряются, потому что сигнал, вызвавший прерывание, сохраняется аппаратурой. Временно задержанное таким способом прерывание называется *отложенным*. Когда прерывания соответствующего класса вновь разрешаются, сигнал опознается и происходит прерывание.

Маскирование прерываний находится под контролем операционной системы и зависит от значения MASK в слове состояния, которое заранее сохраняется в рабочей области каждого прерывания.

Контрольные вопросы

1. Что означают термины «процесс», «задача», «поток»?
2. В каких состояниях в мультипрограммной системе может находиться поток?
3. Какие алгоритмы планирования потоков Вам известны?
4. При каком алгоритме планирования процесс может бесконечно долго находиться в системе, не имея возможности завершиться?
5. Какие из них наиболее подходят для ОС пакетной обработки, системы реального времени, системы разделения времени?
6. Поясните разницу между симметричной архитектурой мультипроцессорной системы и ассиметричной.
7. Может ли процесс в мультипрограммном режиме выполняться быстрее, чем в монопольном?
8. Дайте объяснение вытесняющей и невытесняющей многозадачности.
9. Какие операционные системы являются более надежными: в которых реализована вытесняющая многозадачность или в которых реализована невытесняющая многозадачность?
10. Какие события могут стать причиной перепланировки потоков?
11. Какие типы прерываний Вам известны?
12. Поясните разницу между программным и аппаратным прерываниями.

6. ОПЕРАЦИОННАЯ СИСТЕМА И ПАМЯТЬ

6.1. Типы адресации

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются символьные имена, виртуальные адреса и физические адреса.

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные адреса, называемые иногда математическими, или логическими адресами, вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что начальным адресом программы будет нулевой адрес.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды.

Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством.* Диапазон возможных адресов виртуального пространства у всех процессов является одним и тем же. Например, при использовании 32-разрядных виртуальных адресов этот диапазон задается границами 00000000_{16} и $FFFFFFFF_{16}$. Тем не менее, каждый процесс имеет собственное виртуальное адресное пространство – транслятор присваивает виртуальные адреса переменным и кодам каждой программе независимо.

В разных операционных системах используются разные способы структуризации виртуального адресного пространства. В одних ОС виртуальное адресное пространство процесса подобно физической памяти представлено в виде непрерывной линейной последовательности виртуальных адресов. Такую структуру адресного пространства называют также *плоской* (*flat*). При этом виртуальным адресом является единственное число, представляющее собой смещение относительно начала виртуального адресного пространства. Адрес такого типа называют *линейным виртуальным адресом*.

В других ОС виртуальное адресное пространство делится на части, называемые сегментами (или секциями, или областями, или другими терминами). В этом случае помимо линейного адреса может быть использован виртуальный адрес, представляющий собой *пару* чисел (*n, m*), где *n* определяет сегмент, а *m* – смещение внутри сегмента.

Задачей операционной системы является отображение индивидуальных виртуальных адресных пространств всех одновременно выполняющихся процессов на общую физическую память. При этом ОС отображает либо все виртуальное адресное пространство, либо только определенную его часть. Для пользователя и программиста процедура преобразования виртуальных адресов в физические должна быть максимально прозрачна.

Существуют два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические:

1. Замена виртуальных адресов на физические выполняется один раз для каждого процесса во время начальной загрузки программы в память. Специальная системная программа – *перемещающий загрузчик* – на основании имеющихся у нее исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставленной транслятором об адресно-зависимых элементах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

2. Программа загружается в память в неизменном виде в виртуальных адресах, то есть операнды инструкций и адреса переходов имеют те значения, которые выработал транслятор.

В некоторых случаях (обычно в специализированных системах), когда заранее точно известно, в какой области оперативной памяти будет выполняется программа, транслятор выдает исполняемый код сразу в физических адресах.

Различают два типа виртуального адресного пространства процесса: *максимально возможное* и *назначенное* (*выделенное*).

Максимально возможное виртуальное адресное пространство определяется архитектурой компьютера, на котором работает ОС, и, в частности, разрядностью его схем адресации (32-битная, 64-битная и т.п.). Например, при работе на компьютерах с 32-разрядными процессорами Intel Pentium операционная система может предоставить каждому процессу виртуальное адресное пространство до 4 Гбайт (2^{32}). Но фактически, на практике, процесс использует только часть этого доступного ему виртуального адресного пространства. Максимальный размер виртуального адресного пространства ограничивается только разрядностью адреса, присущей данной архитектуре компьютера, и обычно не совпадает с объемом физической памяти, имеющимся в компьютере.

Назначенное виртуальное адресное пространство представляет собой набор виртуальных адресов, действительно нужных процессу для работы. Эти адреса первоначально назначает программе транслятор на основании текста программы, когда создает кодовый (текстовый) сегмент, а также сегмент данных, с которыми программа работает. Затем

при создании процесса ОС фиксирует назначенное виртуальное адресное пространство в своих системных таблицах.

В ходе своего выполнения процесс может увеличить размер первоначального назначенного ему виртуального адресного пространства, запросив у ОС создания дополнительных сегментов или увеличения размера существующих. В любом случае операционная система обычно следит за корректностью использования процессом виртуальных адресов – процессу не разрешается оперировать с виртуальным адресом, выходящим за пределы назначенных ему сегментов.

Содержимое назначенного процессу виртуального адресного пространства, то есть коды команд, исходные и промежуточные данные, а также результаты вычислений, представляют собой *образ процесса*.

6.2. Алгоритмы распределения памяти

Все алгоритмы распределения памяти разделены на два класса: алгоритмы, в которых используется внешняя память, и алгоритмы, в которых внешняя память не привлекается (рис.6).

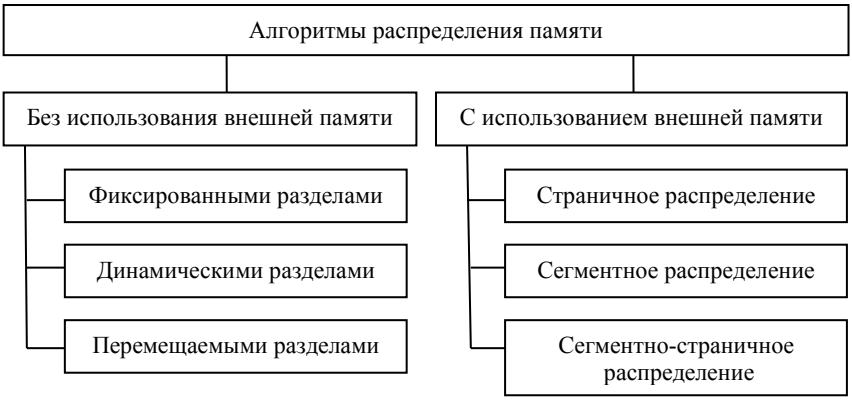

Рис. 6. Методы распределения памяти

6.2.1. Распределение памяти разделами

Распределение памяти фиксированными разделами

Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых *разделами.* Такое разбиение может быть выполнено

вручную оператором во время старта системы или во время ее установки. После этого границы разделов не изменяются. Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу.

Подсистема управления памятью в этом случае выполняет следующие задачи:

– сравнивает объем памяти, требуемый для вновь поступившего процесса, с размерами свободных разделов и выбирает подходящий раздел;

– осуществляет загрузку программы в один из разделов и настройку адресов.

При явной простоте реализации, что является преимуществом данного метода, имеется и существенный недостаток – жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов. Независимо от размера программы она будет занимать весь раздел. Так, например, в системе с тремя разделами невозможно выполнять одновременно более трех процессов, даже если им требуется совсем мало памяти. Кроме того, разбиение памяти на разделы не позволяет выполнять процессы, программы которых не помещаются ни в один из разделов, но для которых было бы достаточно памяти нескольких разделов.

Такой способ управления памятью применялся в ранних мультипрограммных ОС. Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени, в основном благодаря небольшим затратам на реализацию. Детерминированность вычислительного процесса систем реального времени (заранее известен набор выполняемых задач, их требования к памяти, а иногда и моменты запуска) компенсирует недостаточную гибкость данного способа управления памятью.

Распределения памяти динамическими разделами

В этом случае память машины не делится заранее на разделы. Сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнения приложению на этапе создания процесса выделяется вся необходимая ему память (если достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для него не создается). После завершения процесса память освобождается, и на это место может быть загружен другой процесс. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

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

– ведение таблиц свободных и занятых областей, в которых указывается начальные адреса и размеры участков памяти;

– анализ требований к памяти, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения кодов и данных нового процесса (при создании нового процесса);

– загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей;

– после завершения процесса корректировка таблиц свободных и занятых областей.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – фрагментация памяти. *Фрагментация* – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Этот размер настолько мал, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, даже если суммарный объем фрагментов значительно больше требуемого объема памяти.

Распределение памяти динамическими разделами лежит в основе подсистем управления памятью многих мультипрограммных операционных системах 60-70-х годов, в частности такой популярной операционной системы, как OS/360 .

Распределения памяти перемещаемыми разделами

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала единую свободную область. В дополнение к функциям, которые выполняет ОС при распределении памяти динамическими разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется *сжатием.* Сжатие может выполняться либо при каждом завершение процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, а во втором – реже выполняется процедура сжатия.

Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то в данном случае невозможно выполнить настройку адресов с помощью перемещающего загрузчика. Здесь более подходящим оказывается динамическое преобразование адресов.

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

6.2.2. Страничное распределение памяти

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами*. В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемого *физическими страницами* (или блоками, или кадрами).

Размер страницы выбирается равным степени двойки: 512, 1024 и т.д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. Для каждого процесса операционная система создает *таблицу страниц* – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Запись таблицы называется *дескриптором страницы* и включает следующую информацию:

– *номер физической страницы*, в которую загружена данная виртуальная страница;

– *признак присутствия*, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;

– *признак модификации* страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

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

Признаки присутствия, модификации и обращения в большинстве моделей современных процессоров устанавливается аппаратно, схемами процессора при выполнении операции с памятью.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащий требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в оперативной

памяти, то выполняется преобразование виртуального адреса в физический, то есть виртуальный адрес заменяется указанным в записи таблицы физическим адресом. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое *страничное прерывание*. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди процессов, находящихся в состояние готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу (для этого операционная система должна помнить положение вытесненной страницы в страничном файле диска) и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную память, обнуляется ее бит присутствия и анализируется ее признак модификации. Если выталкиваемая страница за время последнего пребывания в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой виртуальной страницы, и никакой записи на диск не производится. Физическая страница объявляется свободной.

Виртуальный адрес при страничном распределении может быть представлен в виде пары (p, s_V), где p – порядковый номер виртуальной страницы процесса (нумерация страниц начинается с 0), а s_V – смещение в пределах виртуальной страницы. Физический адрес также может быть представлен в виде пары (n, s_f), где n – номер физической страницы, а s_f – смещение в пределах физической страницы. Задача подсистемы виртуальной памяти состоит в отображении (p, s_V) в (n, s_f).

Схема преобразования виртуального адреса в физический:

– младшие разряды физического адреса, соответствующие смещению, получаются переносом такого же количества младших разрядов из виртуального адреса;

– старшие разряды физического адреса, соответствующие номеру физической страницы, определяются из таблицы страниц, в которой указывается соответствие виртуальных и физических страниц.

Типичная машинная инструкция требует 3–4 обращений к памяти (выборка команды, извлечение операндов, запись результата). И при каждом обращении происходит либо преобразование виртуального адреса в физический, либо обработка страничного прерывания. Время выполнения этих операций в значительной степени влияет на общую произво-

дительность вычислительной системы, поэтому столь большое внимание разработчиков уделяется оптимизации виртуальной памяти.

Именно для уменьшения времени преобразования адресов во всех процессорах предусмотрен аппаратный механизм получения физического адреса по виртуальному. С той же целью размер страницы выбирается равным степени двойки, благодаря чему двоичная запись адреса легко разделяется на номер страницы и смещение, и в результате в процедуре преобразования адресов более длительная операция сложения заменяется операцией присоединения (конкатенации).

Другим важным фактором, влияющим на производительность системы, является частота страничных прерываний, на которую, в свою очередь, влияет размер страницы и принятые в данной системе правила выбора страниц для выгрузки и загрузки.

6.2.3. Сегментное распределение памяти

При страничной организации виртуальное адресное пространство процесса делится на равные части *механически,* смысловое значение данных не учитывается. В одной странице могут оказаться и коды команд, и инициализируемые переменные, и массив исходных данных программы. Такой подход не позволяет обеспечить дифференцированный доступ к разным частям программы. Разбиение виртуального адресного пространства на «осмысленные» части делает принципиально возможным совместное использование фрагментов программ разными процессами.

Виртуальное адресное пространство процесса делится на части – *сегменты*, размер которых определяется с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями. Максимальный размер сегмента определяется разрядностью виртуального адреса, например при 32-разрядной организации процессора он равен 4 Гбайт. При этом максимально возможное виртуальное адресное пространство процесса представляет собой набор из N виртуальных сегментов, каждый размером по 4 Гбайт. В каждом сегменте виртуальные адреса находятся в диапазоне от 00000000_{16} до $FFFFFFFF_{16}$. Сегменты не упорядочиваются друг относительно друга, так что общего для сегментов линейного виртуального адреса не существует, виртуальный адрес задается парой чисел: номером сегмента и линейным виртуальным адресом внутри сегмента.

При загрузке процесса в оперативную память помещается только часть его сегментов, полная копия виртуального адресного пространства

находится в дисковой памяти. Для каждого загружаемого сегмента операционная система подыскивает непрерывный участок свободной памяти достаточного размера. Смежные в виртуальной памяти сегменты одного процесса могут занимать в оперативной памяти несмежные участки. Если во время выполнения процесса происходит обращение по виртуальному адресу, относящемуся к сегменту, который в данный момент отсутствует в памяти, то происходит прерывание. ОС приостанавливает активный процесс, запускает на выполнение следующий процесс из очереди, а параллельно организует загрузку нужного сегмента с диска. При отсутствии в памяти места, необходимого для загрузки сегмента, операционная система выбирает сегмент на выгрузку, при этом она использует критерии, аналогичные рассмотренным выше критериям выбора страниц при страничном способе управления памятью.

На этапе создания процесса во время загрузки его образа в оперативную память система создает *таблицу сегментов* процесса (аналогичную таблице страниц). В таблице для каждого сегмента указывается: базовый физический адрес сегмента в оперативной памяти; размер сегмента; правила доступа к сегменту; признаки модификации, присутствия и обращения к данному сегменту; некоторая другая информация.

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

Как видно, сегментное распределение памяти имеет очень много общего со страничным распределением. Механизмы преобразования адресов этих двух способов управления памятью тоже весьма схожи, однако в них имеются и существенные отличия, которые являются следствием того, что сегменты в отличие от страниц имеют произвольный размер.

Недостатки сегментного распределения памяти:

1. Нельзя использовать операцию конкатенации, как это делается при страничной организации памяти. Использование операции сложения вместо конкатенации замедляет процедуру преобразования виртуального адреса в физический по сравнению со страничной организацией.

2. Избыточность. При сегментной организации единицей перемещения между памятью и диском является сегмент, имеющий в общем случае объем больший, чем страница. Однако во многих случаях для работы программы вовсе не требуется загружать весь сегмент целиком, достаточно было бы одной или двух страниц. Аналогично при отсутствии свободного места в памяти не стоит выгружать целый сегмент, когда можно обойтись выгрузкой нескольких страниц.

3. Фрагментация, которая возникает из-за непредсказуемости размеров сегментов. В процессе работы системы в памяти образуются не-

большие участки свободной памяти, в которые не может быть загружен ни один сегмент. Суммарный объем, занимаемый фрагментами, может составить существенную часть общей памяти системы, что приводит к ее неэффективному использованию.

Система с сегментной организацией функционирует аналогично системе со страничной организацией: при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический, время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются.

Одним из существенных отличий сегментной организации памяти от страничной является возможность задания дифференцированных прав доступа процесса к его сегментам. Например, один сегмент данных, содержащий исходную информацию для приложения, может иметь права доступа «только чтение», а сегмент данных, представляющий результаты, – «чтение и запись». Это свойство дает принципиальное преимущество сегментной модели памяти над страничной.

6.2.4. Сегментно-страничное распределение памяти

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

Так же как и при сегментной организации памяти, виртуальное адресное пространство процесса разделено на сегменты. Это позволяет определять разные права доступа к разным частям кодов и данных программы. Перемещение данных между памятью и диском осуществляется не сегментами, а страницами. Для этого каждый виртуальный сегмент и физическая память делятся на страницы равного размера, что позволяет более эффективно использовать память, минимизируя фрагментацию.

В большинстве современных реализаций сегментно-страничной организации памяти в отличие от набора виртуальных диапазонов адресов при сегментной организации памяти все виртуальные сегменты образуют одно непрерывное линейное виртуальное адресное пространство.

Координаты байта в виртуальном адресном пространстве при сегментно-страничной организации можно задать двумя способами:

1. Линейным виртуальным адресом, который равен сдвигу данного байта относительно границы общего линейного виртуального пространства.

2. Парой чисел, одно из которых является номером сегмента, а другое – смещением относительно начала сегмента. При этом в отличие от сегментной модели, для однозначного задания виртуального адреса

необходимо указать также начальный виртуальный адрес сегмента с данным номером.

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

Для каждого процесса операционная система создает отдельную таблицу сегментов, в которой содержатся описатели (дескрипторы) всех сегментов процесса Описание сегмента включает назначенные ему права доступа и другие характеристики, аналогичные тем, которые содержатся в дескрипторах сегментов при сегментной организации памяти.

Преобразование виртуального адреса в физический происходит в два этапа:

1. Работает механизм сегментации. Исходный виртуальный адрес, заданный в виде пары (номер сегмента, смещение), преобразуется в линейный виртуальный адрес. Для этого на основании базового адреса таблицы сегментов и номера сегмента вычисляется адрес дескриптора сегмента. Анализируются поля дескриптора и выполняется проверка возможности выполнения заданной операции. Если доступ к сегменту разрешен, то вычисляется линейный виртуальный адрес путем сложения базового адреса сегмента, извлеченного из дескриптора, и смещения, заданного в исходном виртуальном адресе.

2. Работает страничный механизм. Полученный линейный виртуальный адрес преобразуется в искомый физический адрес. В результате преобразования линейный виртуальный адрес представляется в том виде, в котором он используется при страничной организации памяти, а именно в виде пары (номер страницы, смещение в странице). Далее преобразование адреса происходит так же, как при страничной организации: старшие разряды линейного виртуального адреса, содержащие номер виртуальной страницы, заменяются номером физической страницы, взятым из таблицы страниц, а младшие разряды виртуального адреса, содержащие смещение, остаются без изменения.

Управление памятью, основанное на комбинировании сегментного и страничного механизмов, может осуществляться и по другой схеме. Так же как и в предыдущем случае, виртуальное пространство процесса делится на сегменты, а каждый сегмент, в свою очередь, делится на виртуальные страницы. Отличие данной схемы от рассмотренной выше:

1. Виртуальные страницы нумеруются не в пределах всего адресного пространства процесса, а в пределах сегмента. Виртуальный адрес в этом случае выражается тройкой (номер сегмента, номер страницы, смещение в странице).

2. Таблица сегментов состоит из дескрипторов сегментов, которые вместо информации о расположении сегментов в виртуальном адресном пространстве содержат описание расположения таблиц страниц в физической памяти. Таблица страниц содержит дескрипторы страниц, содержимое которых полностью аналогично содержимому ранее описанных дескрипторов страниц.

Для данного метода схема преобразования виртуального адреса в физический будет выглядеть следующим образом:

– по номеру сегмента, заданному в виртуальном адресе, из таблицы сегментов извлекается физический адрес соответствующей таблицы страниц;

– по номеру виртуальной страницы, заданному в виртуальном адресе, из таблицы страниц извлекается дескриптор, в котором указан номер физической страницы;

– к номеру физической страницы присоединяется младшая часть виртуального адреса – смещение.

6.3. Кэш-память

Память вычислительной машины представляет собой набор запоминающих устройств (ЗУ), отличающихся средним временем доступа к данным, объемом и стоимостью хранения одного бита (табл.1).

Таблица 1

Характеристика запоминающих устройств

Тип запоминающих устройств	**Объем памяти**	**Время доступа к данным**
Регистры процессора	Десятки байт	~ 2-3 нс
Быстродействующая память	Десятки-сотни килобайт	~ 5-8 нс
Оперативная память	Десятки мегабайт	~ 10-20 нс
Внешняя память	Десятки гигабайт	~ Десятки мс

Из таблицы видно, что чем больше объем устройства, тем менее быстродействующим оно является. Кроме того, стоимость хранения данных в расчете на один бит также увеличивается с ростом быстродействия устройств. Однако пользователю хотелось бы иметь и недорогую, и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Кэш-память – это способ совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и сто-

имостью хранения данных. Данный способ позволяет уменьшить среднее время доступа к данным и экономить более дорогую быстродействующую память за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ. Кэш-памятью, или кэшем, часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств – «быстрое» ЗУ. «Медленное» ЗУ будем называть ниже основной памятью.

Кэш-память обладает свойством прозрачности для программ и пользователей: ни пользователи, ни программы не принимают никакого участия в перемещении данных из ЗУ одного типа в ЗУ другого типа, все это делается автоматически системными средствами.

Кэширование – это универсальный метод, пригодный для ускорения доступа к оперативной памяти, к диску и к другим видам запоминающих устройств. Если кэширование применяется для уменьшения среднего времени доступа к оперативной памяти, то в качестве кэша используют быстродействующую статическую память. Если кэширование используется системой ввода-вывода для ускорения доступа к данным, хранящимся на диске, то в этом случае роль кэш-памяти выполняют буферы в оперативной памяти, в которых оседают наиболее активно используемые данные. Виртуальную память также можно считать одним из вариантов реализации принципа кэширования данных, при котором оперативная память выступает в роли кэша по отношению к внешней памяти – жесткому диску. В этом случае кэширование используется не для того, чтобы уменьшить время доступа к данным, а для того, чтобы заставить диск частично подменить оперативную память за счет перемещения временно неиспользуемого кода и данных на диск с целью освобождения места для активных процессов. В результате наиболее интенсивно используемые данные остаются в оперативной памяти, остальная же информация хранится в более объемной и менее дорогостоящей внешней памяти.

Принцип действия кэш-памяти

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

– значение элемента данных;

– адрес, который этот элемент данных имеет в основной памяти;

– дополнительную информацию, которая используется для реализации алгоритма замещения данных в кэше и обычно включает признак модификации и признак действительности данных.

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти, чтобы определить, есть ли там нужные данные. Кэш-память не является адресуемой, поэтому поиск

нужных данных осуществляется по содержимому – по взятому из запроса значению поля адреса в оперативной памяти. Далее возможен один из двух вариантов развития событий:

1. Если данные обнаруживаются в кэш-памяти, то есть произошло *кэш-попадание*, они считываются из нее и результат передается источнику запроса;

2. Если нужные данные отсутствуют в кэш-памяти, то есть произошел *кэш-промах*, они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память.

Вероятность обнаружения данных в кэше зависит от разных факторов, таких, например, как объем кэша, объем кэшируемой памяти, алгоритм замещения данных в кэше, особенности выполняемой программы, время ее работы, уровень мультипрограммирования и других особенностей вычислительного процесса. В большинстве реализаций кэш-памяти процент кэш-попаданий оказывается весьма высоким – свыше 90 %. Высокое значение вероятности нахождения данных в кэш-памяти объясняется наличием у данных объективных свойств: пространственной и временной локальности.

Временная локальность. Если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время.

Пространственная локальность. Если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Способы отображения основной памяти на кэш

Алгоритм поиска и алгоритм замещения данных в кэше непосредственно зависят от того, каким образом основная память отображается на кэш-память. Принцип прозрачности требует, чтобы правило отображения основной памяти на кэш-память не зависело от работы программ и пользователей. При кэшировании данных из оперативной памяти широко используются две основные схемы отображения: случайное отображение и детерминированное отображение.

При *случайном* отображении элемент оперативной памяти в общем случае может быть размещен в произвольном месте кэш-памяти. Для того чтобы в дальнейшем можно было найти нужные данные в кэше, они помещаются туда вместе со своим адресом (адресом в оперативной памяти). При каждом запросе к оперативной памяти выполняется поиск в кэше, причем критерием поиска выступает адрес оперативной памяти из запроса. Для кэшей со случайным отображением используется так называемый *ассоциативный поиск,* при котором сравнение выполняется не последовательно с каждой записью кэша, а параллельно со всеми его записями. Признак, по которому выполняется сравнение, называется

тегом. В данном случае тегом является адрес данных в оперативной памяти. Электронная реализация такой схемы приводит к удорожанию памяти, причем стоимость существенно возрастает с увеличением объема запоминающего устройства. Поэтому ассоциативная кэш-память используется в тех случаях, когда для обеспечения высокого процента попадания достаточно небольшого объема памяти.

В кэшах, построенных на основе случайного отображения, вытеснение старых данных происходит только в том случае, когда вся кэш-память заполнена и нет свободного места. Выбор данных на выгрузку осуществляется среди всех записей кэша.

Детерминированный способ отображения предполагает, что любой элемент основной памяти всегда отображается в одно и то же место кеш-памяти. В этом случае кэш-память разделена на строки, каждая из которых предназначена для хранения одной записи об этом элементе данных и имеет свой номер. Между номерами строк кэш памяти и адресами оперативной памяти устанавливается соответствие «один ко многим»: одному номеру строки соответствует несколько адресов оперативной памяти.

В качестве отображающей функции может использоваться простое выделение нескольких разрядов из адреса оперативной памяти, которые интерпретируются как номер строки кэш-памяти. Такое отображение называется *прямым*.

Во многих современных процессорах кэш-память строится на основе сочетания этих двух подходов, что позволяет найти компромисс между сравнительно низкой стоимостью кэша с прямым отображением и интеллектуальностью алгоритмов замещения в кэше со случайным отображением. При смешанном подходе произвольный адрес оперативной памяти отображается не на один адрес кэш-памяти (как это характерно для прямого отображения) и не на любой адрес кэш-памяти (как это делается при случайном отображении), а на некоторую группу адресов. Все группы пронумерованы. Поиск в кэше осуществляется вначале по номеру группы, полученному из адреса оперативной памяти из запроса, а затем в пределах группы путем ассоциативного просмотра всех записей в группе на предмет совпадения старших частей оперативной памяти. Таким образом, в данном способе комбинируется прямое отображение на группу и случайное отображение в пределах группы.

Контрольные вопросы

1. Чем ограничивается максимальный размер виртуального адресного пространства, доступного приложению?

2. В каких случаях транслятор создает объектный код программы не в виртуальных, а в физических адресах?

3. Что такое виртуальная память?

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

5. Почему размер страницы выбирается равным степени двойки? Можно ли принять такое же ограничение для сегмента?

6. На что влияет размер страницы? Каковы преимущества и недостатки большого размера страницы?

7. Где хранятся таблицы страниц и таблицы сегментов?

8. Чем определяется количество таблиц сегментов, имеющихся в операционной системе в произвольный момент времени?

9. Какие характеристики содержит таблица сегментов и таблица страниц при сегментно-страничной организации памяти?

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

11. Почему загрузка и выгрузка данных из кэш-памяти производится блоками?

7. СОВРЕМЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

7.1. Операционные системы семейства Windows

7.1.1. Основные характеристики

Основными характеристиками ОС Windows являются:

1. 32-разрядная архитектура означает, что операции над 32-разрядным данными выполняются заметно быстрее, чем в 16-разрядных операционных системах, поскольку в них требуется программная реализация над 32-разрядными данными.

2. Многозадачность – способность «одновременно» (параллельно) выполнять несколько программ.

3. Многопоточность операционной системы означает, что работающие программы (процессы) могут разделяться на несколько частей, самостоятельно претендующих на процессорное время. Это обеспечивает одновременное выполнение программой нескольких не связанных друг с другом опера ций. Например, в текстовом процессоре могут одновременно выполняться автоматическая проверка орфографии и редактирование документа.

4. Графический пользовательский интерфейс обеспечивает удобства в запуске и переключении приложений (программ Windows). Основными компонентами пользовательского интерфейса являются рабочий

стол и панель задач (обеспечивает запуск и переключение приложений). На рабочем столе размещены графические объекты, соответствующие приложениям, документам, сетевым устройствам. Каждый графический объект имеет поименованный ярлычок. С помощью мыши, ярлычков, главного меню и панели задач пользователь может легко запускать, переключать и управлять приложениями (программами).

5. Подключение новых периферийных устройств по технологии Plug and Play. Система самостоятельно создает и изменяет файлы конфигурации, распознает конкретное техническое устройство, производит его автонастройку. Подобная технология получила название Plug and Play – «включай и работай», т. е. эта технология ориентирована на поддержку любого типа устройств, включая мониторы, видеоплаты, принтеры, звуковые карты, модемы, приводы CD, контроллеры магнитных дисков. При ее использовании обеспечиваются следующие вспомогательные функции: распознавание устройств для установки и настройки, динамическое изменение состояния системы, интеграция драйверов устройств, системных компонентов и пользовательского интерфейса.

6. Пользование виртуальной памяти. Виртуальная память – расширение адресного пространства задачи, полученное за счет использования части внешней памяти.

7. Совместимость с ранее созданным программным обеспечением. Под совместимостью с программным обеспечением понимают способность операционной системы исполнять программные продукты, созданные в другой операционной системе.

8. Наличие коммуникационных программных средств. Важнейшим направлением развития Windows является включение в ее структуру используемых и специальных программных средств для поддержки различных коммуникаций и компьютерных сетей. Сетевые средства операционной системы Windows позволяют: обеспечить передачу данных между двумя соединенными кабелями; организовать электронную почту в локальной и глобальной сети; выполнить факсимильную передачу; обменяться файлами с удаленным компьютером и подключиться к глобальной сети.

9. Наличие средств мультимедиа. Система Windows обеспечивает интерактивную работу с видео и аудио программами при помощи специальных аппаратных и программных средств. Атрибутом мультимедиакомпьютера являются звуковая плата, которая обеспечивает преобразование звука в компьютерную форму и обратно, и видеоплата, которая преобразует видеоинформацию в компьютерную форму и обратно. К звуковой плате подключаются различные акустические системы.

10. Интеграция с глобальной сетью Интернет. Операционная система Windows содержит прикладную программу, позволяющую полу-

чать из Internet различные документы, просматривать и редактировать их содержимое.

11. Надежность и качество управления.

12. Поддержка длинных имен файлов. В Windows имена файлов могут иметь длину до 255 символов.

Сервис операционной системы включает в свой состав подсистему поддержки самонастраивающейся аппаратуры Plug and Play, а также набор различных прикладных функций, например выдачи текущих даты и времени. Подсистема управления виртуальной машины реализует все действия по управлению задачами, управлению памятью, загрузкой и завершением программ, а также обслуживанием виртуальных драйверов устройств. Драйверы устройств могут быть самыми разнообразными, в том числе драйверами реального режима или виртуальными драйверами внешних устройств. Драйверы внешних устройств позволяют нескольким приложениям одновременно использовать одно устройство, например экран монитора.

В операционную систему Windows входит ограниченный набор прикладных и служебных программ (см. п.7.1.2 «Служебные приложения»), с помощью которых можно решать некоторые простейшие повседневные задачи, пока на компьютере не установлены более мощные программы средства. Такие программы, входящие в поставку Windows, называются стандартными приложениями. Стандартные приложения – это комплекс простых программ, поставляемых вместе с ОС Windows. К прикладным программам относятся простой текстовый редактор «Блокнот», графический редактор Paint, простой текстовый процессор WordPad.

Кроме, прикладных и служебных программ ОС Windows содержит стандартные средства мультимедиа: регулятор громкости; лазерный проигрыватель; универсальный проигрыватель; программа звукозаписи.

7.1.2. Служебные приложения

Служебные приложения предназначены для обслуживания персонального компьютера и самой операционной системы. Они позволяют находить и устранять дефекты файловой системы, оптимизировать настройки программного аппаратного обеспечения, а также автоматизировать некоторые рутинные операции, связанные с обслуживанием компьютера.

В главном меню служебные приложения Windows сосредоточены в категории Пуск/ Программы/ Стандартные/ Служебные. Они поставляются в составе ОС и устанавливаются вместе с ней (полностью или выборочно). Кратко охарактеризуем основные служебные приложения.

1. *Архивация данных*

Программа предназначена для автоматизации регулярного резервного копирования наиболее ценных данных на внешнем носители. Приложение ориентированно на работу с внешними накопителями большой емкости. При использовании ненадежных носителей следует создавать не менее двух резервных копий.

Программа Архивации данных позволяет:

– создавать задания на архивацию, при этом указываются папки, содержимое которых подлежит резервному копированию;

– выполнять полную или частичную архивацию;

– в случае утраты данных выполнять их восстановление из резервной копии.

При организации резервного копирования следует иметь в виду следующие принципы:

а) архивации подлежат только данные, но не программы;

б) резервное копирование выполняется тем чаще, чем чаще обновляются данные и чем выше их ценность;

в) резервные копии данных хранят отдельно от компьютера.

2. *Буфер обмена*

Приложение предназначено для просмотра текущего содержания буфера обмена Windows.

3. *Дефрагментация диска*

Приложение предназначено для повышения эффективности работы жесткого диска путем устранения фрагментированности файловой структуры.

Наименьшая единица хранения данных на диске – кластер. Если свободного места на диске достаточно, то файлы записываются так, что кластеры, в которые происходит запись, располагаются последовательно. В этом случае обращение к файлу происходит быстро, так как затраты времени на поиск очередного кластера минимальны. Если же диск заполнен до отказа, запись на него возможна только после освобождения некоторых кластеров, путем удаления файлов. При этом свободные области, образующиеся на диске, не составляют одну большую непрерывную область. При попытке записать длинный файл на диск, файл делится на фрагменты, которые записываются туда, где им нашлось место. Длительная работа с заполненным жестким диском приводит к постепенному увеличению фрагментированности файлов и значительному замедлению работы.

Программа Дефрагментация диска выполняет перекомпоновку файлов таким образом, что длинные файлы собираются из коротких фрагментов. В результате доступ к файлам заметно упрощается и эффективность работы компьютера возрастает.

4. *Индикатор системных ресурсов*

После запуска этого приложения на панели индикации устанавливается небольшой значок, посредством которого можно получить сведения о состоянии системных ресурсов.

Ресурсами в данном случае называются специальные модули памяти, предназначенные для обслуживания многозадачного режима работы. Всякой запущенной программе выдается небольшая порция этих модулей. По завершению своей работы приложение должно освободить ресурсы, но некорректно работающие (или устаревшие) программы могут этого не сделать. В таком случае возникает явление «утечки ресурсов», и программа может «зависнуть». Характерным проявлением явления «утечки ресурсов» служат сообщения от ОС о недостатке оперативной памяти, в то время как на самом деле ее вполне достаточно. Для контроля за подобными нештатными ситуациями используют Индикатор системных ресурсов.

5. *Проверка диска*

Программа позволяет выявить логические ошибки в файловой структуре (стандартная проверка), а также физические ошибки, связанные с дефектами поверхности жесткого диска (полная проверка). Стандартную проверку рекомендуется проводить после каждого сбоя в работе компьютера, а полную – примерно два раза в год.

6. *Сведения о системе*

Это специальный пакет программных средств, собирающих сведения о настройке операционной системы Windows, ее приложений и оборудования компьютерной системы. Средства этого пакета предназначены для специалистов, выполняющих ремонтно-восстановительные работы.

7. *Сжатие данных*

Программа предназначена для повышения плотности записи данных на жесткий диск. В ее основе лежит принцип устранения избыточности информации.

8. *Системный монитор*

Программа предназначена для визуального или протокольного наблюдения за функционированием компьютера и ОС. Она позволяет контролировать загрузку процессора, распределение оперативной памяти, обмен данными между дисками и другие параметры вычислительной системы. Результаты наблюдения можно отобразить на экране в виде графиков или записывать в протокольный файл. Исследование компьютера с помощью программы Системный Монитор позволяет находить «узкие места» в производительности компьютерной системы, сравнивать между собой варианты настройки аппаратных и программных средств.

7.1.3. Настройка операционной системы

Цель настройки состоит в создании условий для эффективной работы путем автоматизации операций и создания комфортной рабочей среды. Основные настраиваемые объекты – средства управления и оформления.

Средствами настройки является:

– специальная папка Панель управления;

– контекстные меню объектов Windows;

– элементы управления диалоговых окон операционной системы и ее приложений;

Кратко опишем возможности настройки операционной системы Windows.

1. Настройка средств ввода-вывода данных:

– настройка клавиатуры (можно настраивать темп повтора символов, частоту мерцания курсора и другие возможности);

– настройка мыши (назначение левой или правой кнопке функции основной кнопки, выбор формы указателя мыши, управление чувствительностью мыши и др.);

– настройка системы управления операционной системой (мы рассматривали так называемый классический стиль: объекты выделяют одним щелчком, открывают двумя щелчками; существует стиль Web, характерный для работы в Интернете и другие стили).

2. Настойка элементов оформления Windows:

– настройка фона рабочего стола;

– настройка экранной заставки;

– настройка оформления элементов управления Windows;

– настройка параметров экрана (величина экранного размещения, и величина цветового размещения);

– настройка свойств видеоадаптера и монитора;

– настройка звуковых схем (звуковой схемой называется именованная совокупность настроек, связанных с назначением определенным событиям определенных звуков);

– оформление Windows темами Рабочего стола (каждая тема представляет собой комплексный элемент оформления, в который входят: схема оформления, схема указателей, звуковая схема, фоновый рисунок, заставка экрана и коллекция значков Рабочего стола – все элементы художественно согласованы в рамках одной темы).

3. Настройка элементов управления Windows:

– настройка Панели задач – можно изменить ее расположение вдоль любой кромки экрана (исходное состояние – нижняя кромка); настроить размер Панели задач (предельный размер – половина экрана);

изменить свойства Панели задач (например, добавить другие инструментальные панели (панель адресов Интернета или др.));

– настройка Главного меню;

– настройка свойств Корзины (например, ее емкость – измеряется в процентах от емкости соответствующих дисков);

– настройка свойств окон папок (можно настраивать режим отображения скрытых и системных объектов, способ обзора вложенных папок: все вложенные папки открываются в одном и том же окне или для каждой очередной вложенной папки открывается новое окно).

4. Настройка средств автоматизации Windows:

– автоматический запуск приложений;

– настройка свойств типов файлов (регистрация типов файлов в ОС осуществляется по расширению имени файла); можно изменить значок, связанный с данным типом файлов, создать новое действие, которое возможно с данным типом файлов, удалить действие или назначить его основным и др.;

– автоматизация очистки жесткого диска (состав папок, подлежащих очистке, задается заранее установкой соответствующих флажков);

– запуск приложений по расписанию (можно выбрать приложение и назначить расписание его запуска с указанием даты и времени, а также периодичность последующих запусков);

– обслуживание компьютера в автоматическом режиме;

– автоматизация поисковых операций (поиск определенных файлов, документов);

5. Настройка шрифтов.

ОС Windows позволяет работать с двумя классами шрифтов – растровые и векторные. Растровые – символы образуются как комбинации точек в матрице заданного размера. Векторные – символы представляют собой криволинейные контуры, составные элементы которых описываются математическими формулами. Операционная система Windows позволяет использовать различные шрифтовые наборы, устанавливать новые шрифты и удалять, обеспечивает единство принципов применения шрифтов в самых разнообразных приложениях.

6. Прочие настройки Windows:

– настройка системных часов и системного календаря (в полном соответствии с текущими настройками при сохранении любого файла вместе с ним сохраняются данные о дате и времени создания или последнего изменения);

– настройка национальных стандартов и форматов (например, различия, связанные с форматами записи чисел, дат, времени, денежных сумм и т.п.).

7.1.4. Эволюция ОС семейства Windows
(учебная проблема)

Контрольные вопросы

1. Почему Б.Гейтс решил заняться созданием графической среды, которая послужила бы стандартной платформой для разработчиков, вместо того, чтобы как все делать интегрированные среды?

2. Какой первоначально представлялась ОС Windows?

3. Каковы требования к компьютеру, на котором работала Windows 1.0?

4. Какие проблемы удалось решить, введя поддержку расширенной памяти (LIM) в Windows 1.03?

5. Чем Windows 2.0 лучше операционной системы Windows 1.0?

6. Для чего понадобился протокол обмена данными (DDE)?

7. Какие недостатки остались в Windows 2.0?

8. Какие улучшения можно наблюдать в Windows 3.0?

9. В чем проявляется псевдо многозначность?

10. Почему Windows 3.0 пользовалась успехом?

11.Каковы недостатки операционной системы Windows 3.0?

12. Версия 3.1 планировалась лишь как небольшое улучшение версии 3.0. Почему Windows 3.1 превратилась в самостоятельный проект?

13. Какие улучшения в интерфейсе и архитектуре Windows 95?

14. Для чего разрабатывалась Windows NT?

15. Каковы нововведения в Windows NT по сравнению с предыдущими версиями Windows?

16. Как изменился интерфейс в Windows NT 4.0?

17. Каковы плюсы и минусы Windows NT 4.0 по сравнению с Windows 95?

18. В чем проявлялись удобства Windows 98?

19. Какие утилиты и служебные программы имеются в Windows 98? Для чего они предназначены?

20. За счет чего удалось повысить быстродействие в ОС Windows 98?

21. Какие полезные функции появились вместе с выходом Windows 98?

22. Что представляют собой средства устранения неполадок? Как получить доступ к ним?

23. На базе чего была создана операционная система Windows Me?

24. Что нового в Windows Me по сравнению с Windows 98? Какие недостатки Windows 98 удалось устранить?

25. Каковы недостатки у Windows Me?

26. Что нового в Windows 2000 по сравнению с предыдущими версиями Windows, в частности с Windows NT 4.0?

27. За счет чего программисты Microsoft добились большей стабильности в работе Windows 2000 по сравнению с Windows 98?

28. Каковы системные требования Windows 2000?

29. Что нового в Windows XP по сравнению с предшествующими версиями Windows?

30. Чем отличаются Windows XP Professional и Windows XP Home Edition?

31. Достоинства и недостатки Windows XP (Professional и Home Edition по отдельности).

32. Каковы системные требования Windows XP?

33. Каковы особенности различных версий операционной системы Windows Vista?

34. Составьте таблицу сравнительных характеристик различных версий операционных систем семейства Windows. Сделайте соответствующие выводы.

7.2. Операционные системы семейства UNIX

(учебная проблема)

Контрольные вопросы

1. На каком языке была написана UNIX? Продолжается ли эта традиция?

2. Какой компанией была разработана UNIX?

3. Какую структуру имеет файловая система UNIX?

4. Охарактеризуйте концепцию виртуальной памяти.

5. Зачем нужен сигнальный механизм в UNIX и как он работает? Виды сигналов. Как UNIX реагирует на различные сигналы?

6. Как осуществляется работа пользователей в UNIX? Как осуществляется доступ к файлам?

7. Каким в UNIX представляется файл? Каков механизм работы с файлами?

8. Какую функцию выполняет буфер при работе с файлами? Как он это делает?

9. Что понимается под потоком в UNIX? Виды потоков.

10. На чем основывается интерфейс пользователя UNIX? Как выглядит и работает графический интерфейс UNIX?

11. Как ядро ОС UNIX идентифицирует пользователя?

12. В чем отличия привилегированного пользователя от обычного?

13. Какой язык является в системе UNIX стандартным? Какие в UNIX есть встроенные функции, удобные для пользователей и программистов?

14. В чем заключаются функции ядра?

15. Почему о системе UNIX можно сказать, что она мобильна?

16. Почему был закрыт проект Multics, предшественник UNIX?

17. Что послужило причиной популярности UNIX?

18. Может ли пользователь изменить имя и пароль?

19. Из чего состоит команда для интерпретатора?

20. Из каких основных компонентов состоит UNIX?

21. Для чего нужны специальные файлы?

22. Как осуществляется подключение к UNIX?

23. Для какой категории пользователей предпочтительнее выбрать своей ОС именно UNIX?

24. Можно ли UNIX назвать идеальной ОС, или у нее есть недостатки?

25. Для чего нужны редакторы VI и ED?

26. Системные требования UNIX.

27. Преимущества и недостатки UNIX в сравнении с конкурентами.

28. Что представляет собой ОС Linux? Системные требования Linux.

29. История Linux. Что связывает Linux и UNIX?

30. Системные характеристики Linux.

31. Процесс установки Linux.

32. Какие проблемы (ошибки) могут возникнуть во время установки Linux? Как их устранить?

33. Какие проблемы могут возникнуть после установки Linux? Как их устранить?

34. Преимущества и недостатки Linux в сравнении с конкурентами.

7.3. Сетевые операционные системы

(учебная проблема)

Контрольные вопросы

1. Какие из утверждений верны:

а) «сетевая операционная система» – это совокупность операционных систем всех компьютеров сети;

б) «сетевая операционная система» – это операционная система отдельного компьютера, способного работать в сети;

в) «сетевая операционная система» – это набор сетевых служб, выполненный в виде оболочки?

2. Что понимается под сетевой операционной системой в широком и узком смыслах?

3. Какой минимум функциональных возможностей надо добавить к локальной ОС, чтобы она стала сетевой?

4. Перечислите основные сетевые службы. Какие из них, как правило, встроены в операционную систему?

5. Из чего состоит сетевая ОС (ее структура)? Для чего предназначена каждая ее часть (схемы работы клиентской и серверной частей)?

6. Какие из утверждений верны:

а) редиректор – клиентская часть сетевой службы;

б) редиректор – модуль, входящий в состав клиентской части сетевой службы, распознающий и перенаправляющий запросы к нужному сетевому серверу или локальной ОС?

7. Поясните значение следующих терминов применительно к сетевым ОС: «сервис», «сервер», «клиент», «служба», «оболочка», «услуга», «редиректор». Какие из них употребляются как синонимы?

8. Может ли сетевая оболочка работать над сетевой ОС?

9. В каких случаях может оказаться полезным наличие сразу нескольких серверных (клиентских) частей файловых служб?

10. Классы сетей (одноранговые и двухранговые). Чем они отличаются?

11. Объясните, как работают одноранговые и двухранговые сети (схемы их работы)? В чем проявляется функциональная и аппаратная несимметричность в каждом классе?

11. Какие из следующих утверждений верны:

а) ОС выделенного сервера никогда не содержит клиентских частей сетевых служб;

б) в одноранговых ОС всегда имеются и клиентские, и серверные части сетевых служб;

в) в сетях с выделенными серверами могут поддерживаться одноранговые связи?

12. Может ли выделенный сервер обращаться с запросами к ресурсам клиентских станций?

13. Приведите примеры одноранговых ОС и ОС с выделенным сервером.

14. Что понимается под нитью в сетевых ОС?

15. Классификация сетей. Назначения сетей каждого вида.

16. Признаки корпоративных ОС. Критерии для выбора ОС масштаба предприятия.

7.4. Требования к современным операционным системам

Главным требованием, предъявляемым к операционной системе, является выполнение ею основных функций: эффективно управлять ресурсами, обеспечивать удобный интерфейс для пользователя и прикладных программ, поддерживать мультипрограммную обработку, виртуальную память, свопинг, многооконный графический интерфейс пользователя и т.д. Кроме этих требований функциональной полноты к операционным системам предъявляются не менее важные эксплуатационные требования, представленные на рисунке 7.

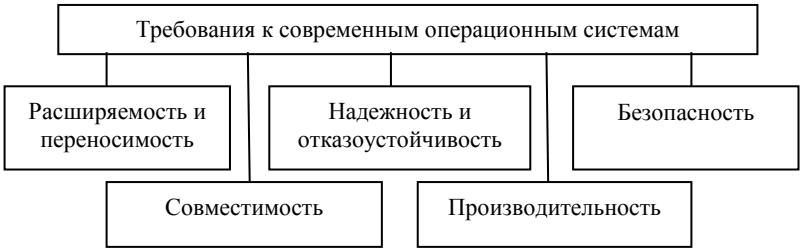

Рис.7. Эксплуатационные требования, предъявляемые к ОС

Расширяемость. В то время как аппаратная часть компьютера устаревает за несколько лет, полезная жизнь операционных систем может измеряться десятилетиями (например, ОС UNIX). Операционные системы всегда изменяются со временем эволюционно, и эти изменения более значимы, чем изменения аппаратных средств. Изменения ОС обычно заключаются в приобретении ею новых свойств, например поддержка новых типов внешних устройств, новых технологий. Если код операционной системы написан таким образом, что дополнения и изменения могут вноситься без нарушения целостности системы, то такую ОС называют расширяемой.

Расширяемость достигается разными способами:

1) За счет модульной структуры ОС, при которой программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс. Новые компоненты добавляются в ОС модульным путем и выполняют свои функции, используя существующие интерфейсы.

2) Использование объектов, которые позволяют единообразно управлять системными ресурсами. Объекты – это абстрактные типы дан-

ных. Над ними можно производить лишь те действия, которые предусмотрены специальным набором объектных функций.

3) Структурирование операционных систем по типу «клиент-сервер», используя микроядерную технологию (см.п.3.5 «Микроядерная архитектура»).

4) Использование средств вызова удаленных процедур, которые можно добавить в любой компьютер сети. Они сразу же поступают в распоряжение прикладных программ на других машинах сети.

5) За счет загрузки новых драйверов, которые добавляются в систему во время её работы.

Переносимость. Код операционной системы должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа на аппаратную платформу другого типа (см. п.3.4.3. «Переносимость операционной системы»). Переносимые ОС имеют несколько вариантов реализации для разных платформ, такое свойство ОС называют также *многоплатформенностью.* Требование переносимости тесно связано с требованием расширяемости.

Совместимость. Существует несколько популярных операционных систем (разновидности UNIX, Windows NT,), для которых наработан широкий спектр приложений. Некоторые из них пользуются огромной популярностью. Поэтому для пользователя, переходящего по каким-либо причинам с одной операционной системы на другую, необходима возможность запуска в новой операционной системе привычного приложения. Если ОС имеет средства для выполнения прикладных программ, написанных для других операционных систем, или для более ранних версий данной ОС, то она обладает совместимостью с этими операционными системами. Рассматривают совместимость на двух уровнях:

– на уровне двоичных кодов;

– на уровне исходных текстов.

Именно двоичная совместимость имеет практическое значение для конечных пользователей, так как благодаря ей один и тот же коммерческий продукт можно использовать в различных ОС и на различных машинах. Совместимость на уровне исходных текстов важна для разработчиков приложений. Более детально вопрос об уровнях совместимости рассмотрен в п.3.6. «Совместимость и множественные прикладные среды».

Понятие совместимости включает также поддержку пользовательских интерфейсов других операционных систем.

Надежность и *отказоустойчивость.* Система должна быть защищена как от внутренних, так и от внешних ошибок, сбоев и отказов. Ее действия должны быть всегда предсказуемыми, а приложения не должны наносить вред ОС. Надежность и отказоустойчивость операционной системы прежде всего определяются архитектурными решениями, поло-

женными в ее основу, а также качеством ее реализации (отлаженностью кода). Кроме того, важно, включает ли ОС программную поддержку аппаратных средств обеспечения отказоустойчивости, таких, например, как дисковые массивы или источники бесперебойного питания.

Производительность. Операционная система должна обладать настолько хорошим быстродействием и временем реакции, насколько это позволяет аппаратная платформа. На производительность ОС влияет много факторов, среди которых основными являются архитектура ОС, многообразие функций, качество программирования кода, возможность исполнения ОС на высокопроизводительной (многопроцессорной) платформе.

Безопасность. Современная ОС должна защищать данные и другие ресурсы вычислительной системы от несанкционированного доступа. Чтобы ОС отвечала требованию безопасности, она должна как минимум иметь в своем составе средства аутентификации – определения легальности пользователей, авторизации – предоставления легальным пользователям дифференцированных прав доступа к ресурсам, аудита – фиксации всех «подозрительных» для безопасности системы событий.

Свойство безопасности особенно важно для сетевых ОС. В таких операционных системах к задаче контроля доступа добавляется задача защиты данных, передаваемых по сети.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Microsoft Windows XP : Home Edition и Professional. – СПб.: БХВ-Петербург, 2005. – 624 с.
2. Бэкон Д., Харрис Е. Операционные системы параллельные и распределенные. – СПб.: Питер; Киев: Издательская группа BHV, 2006. – 800 с.
3. Гладышева М.М., Романов П.Ю. Моделирование системы формирования исследовательских умений будущих инженеров-программистов // Вестник Челябинского государственного педагогического университета. Челябинск: ЧГПУ, 2007. – № 6. – С. 150–161.
4. Гордеев А.В. Операционные системы. – СПб.: Питер, 2005.– 415 с.
5. Гостев, И. М. Операционные системы : учебник и практикум для вузов / И. М. Гостев. – 2-е изд., испр. и доп. – Москва : Издательство Юрайт, 2020. – 164 с. – (Высшее образование). – ISBN 978-5-534-04520-8. – Текст : электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/451231 (дата обращения: 22.09.2020).
6. Давыдов А.П., Злыднева Т.П. Методы математической физики. Параболические уравнения. Курс лекций: учебное пособие. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2019. – 108 с.
7. Дейл Харви. Операционные системы. Распределенные системы, сети, безопасность. – М.: БИНОМ-ПРЕСС, 2007. – 704 с.
8. Злыднева Т.П. Учебно-исследовательская деятельность как условие качественной подготовки специалиста // Вестник педагогических инноваций. – 2005. – № 3. – С. 90–97.
9. Злыднева Т.П. Теоретические подходы к организации исследовательской деятельности студентов // Современные проблемы науки и образования : тезисы докладов XLIII внутривузовской научной конференции преподавателей МаГУ. – Магнитогорск, 2005. – С. 278–279.
10. Злыднева Т.П. К вопросу об эффективности подготовки специалистов в области информатики // Актуальные проблемы информатики и информационных технологий: материалы II Междунар. научно-практической конференции. – Тамбов, 2005. – С. 32–35.
11. Злыднева Т.П. Исследовательская деятельность студентов как педагогическая система // Наука-вуз-школа: сб. науч. тр. молодых исследователей. – Магнитогорск, 2005. – Вып. 10. – С. 38–42.
12. Злыднева Т.П. Личностно-ориентированный подход к организации исследовательской деятельности студентов // Современные технологии образования: сб. науч. тр. – Магнитогорск, 2005. – С. 63–68.

13. Злыднева Т.П. Профессиональная направленность исследовательской деятельности студентов // Педагогические и философские аспекты образования: сб. науч. тр. – Магнитогорск, 2005. – Вып. 2. – С. 38–42.
14. Злыднева Т.П. Педагогические условия эффективности организации исследовательской деятельности студентов вузов // Проблемы истории, филологии, культуры. – 2006. – Вып.17. – С. 305–311.
15. Злыднева Т.П. Модель системы организации исследовательской деятельности студентов // Психология и педагогика современного образования в России: материалы междунар. научно-практической конференции – Пенза, 2006. – С. 99–102.
16. Злыднева Т.П. Обучение студентов исследовательской деятельности в рамках дисциплин информатики // Фундаментальные науки и образование: материалы Всероссийской научно-практической конференции. – Бийск, 2006. – С. 289–293.
17. Злыднева Т.П. Моделирование системы организации исследовательской деятельности студентов вузов в процессе профессиональной подготовки // Вестник Челябинского государственного педагогического университета. – 2006. – № 5. – С. 22–30.
18. Злыднева Т.П. Организация исследовательской деятельности студентов университета в процессе профессиональной подготовки: дис… канд. пед. наук: 13.00.08 – Магнитогорск, 2006. – 204 с.
19. Злыднева Т.П. Организация исследовательской деятельности студентов университета в процессе профессиональной подготовки: автореф. дис… канд. пед. наук – Магнитогорск: Изд-во Магнитогорского государственного университета, 2006. – 24 с.
20. Злыднева Т.П. Методика организации исследовательской деятельности студентов в процессе изучения дисциплин информатики // Университетское образование: материалы междунар. Научно-методической конференции – Пенза, 2007. – С.241–243.
21. Злыднева Т.П. Проблема качественной подготовки специалистов и возможные пути её решения // Педагогические аспекты математического образования: сб. науч. тр. – Магнитогорск, 2008. – Вып. 5. – С. 58–62.
22. Злыднева Т.П. К вопросу об организации учебно-исследовательской деятельности магистрантов // Развитие науки и образования в современном мире: сб. науч. трудов Междунар. научно-практической конференции, часть 4. – М.: «АР-Консалт», 2015. – С. 85–88.
23. Злыднева Т.П. Роль учебно-исследовательской деятельности студентов в реализации ФГОС нового поколения // Физико-математические науки и образование: сборник трудов участников

Всероссийской научно-практической конференции. – Магнитогорск, 2012. – С. 22–24.
24. Злыднева Т.П. Учебно-исследовательская деятельность студентов как необходимая составляющая процесса формирования профессиональных компетенций // Современные проблемы науки и образования: материалы L внутривузовской научной конференции преподавателей МаГУ. – Магнитогорск, 2012. – С. 233–234.
25. Злыднева Т.П. Возможные пути успешной реализации ФГОС ВПО третьего поколения // Педагогические аспекты математического образования: сб. науч. тр. – Магнитогорск, 2012. – Вып. 9. – С. 33–36.
26. Злыднева Т.П. Опытно-экспериментальная работа по организации исследовательской деятельности студентов в процессе изучения дисциплин информатики // Психология и педагогика: пути и методы развития: сб. статей III Междунар. научно-практической конференции – Пенза, 2011. – С. 53–57.
27. Злыднева Т.П. Проблемный подход в изучении дисциплин информатики // Актуальные проблемы информатики и информационных технологий: сб. науч. трудов XIII Междунар. научно-практической конференции. – Тамбов, 2009. – С. 37–41.
28. Злыднева Т.П. История прикладной математики и информатики. Часть 1. История математики: учеб. пособие / Т.П. Злыднева. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2014. – 89 с.
29. Злыднева Т.П. История прикладной математики и информатики. Часть 2. История информатики: учеб. пособие / Т.П. Злыднева – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2014. – 71 с.
30. Злыднева Т.П. Язык С: лабораторный практикум по программированию для студентов специальности «Прикладная математика и информатика» / Т.П. Злыднева. – Магнитогорск: МаГУ, 2005. – 74 с.
31. Злыднева Т.П. Операционные системы. Проблемно-информационный курс : методические рекомендации по изучению курса «Системное и прикладное программное обеспечение» / Т.П. Злыднева. – Магнитогорск: МаГУ, 2007. – 48 с.
32. Злыднева Т.П. Программирование на языке С: лабораторный практикум для студентов специальности «Прикладная математика и информатика» / Т.П. Злыднева. – Магнитогорск: МаГУ, 2008. – 72 с.
33. Злыднева Т.П. Применение методики формирования профессиональных компетенций на различных ступенях высшего образования // Актуальные проблемы гуманитарных и естественных наук. – 2015. – №11. – С. 75–77.

34. Злыднева Т.П. Введение в операционные системы. Проблемно-информационный курс: учеб. пособие. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2015. – 99 с.
35. Злыднева Т.П. Введение в базы данных: учеб. пособие. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2017. – 71 с.
36. Злыднева Т.П. Основы работы с базами данных: практикум. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2018. – 66 с.
37. Злыднева Т.П. Математика и информатика: от истоков до современности: учебное пособие. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2018. – 150 с.
38. Злыднева Т.П. Базы данных: основы теории и проектирования: учебное пособие. – Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2019. – 129 с.
39. Злыднева Т.П. Организация исследовательской деятельности при обучении студентов Web-программированию // Актуальные проблемы современной науки, техники и образования: тезисы докладов 77-й международной научно-технической конференции. Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, – 2019. –Т.2. – С. 67.
40. Злыднева Т.П. Технология проектного обучения Web-программированию в рамках компетентностного подхода. // Журнал естественнонаучных исследований. М.: ИНФРА-М, – 2018. – Т. 3. – № 1. – С. 23–31.
41. Злыднева Т.П. Роль информационных технологий в формировании профессиональных компетенций. / Т.П. Злыднева // Информационные технологии в науке, управлении, социальной сфере и медицине: сборник научных трудов III Международной научной конференции: в 2 частях. Под редакцией: О.Г. Берестневой, О.М. Гергет, Т.А. Гладковой; Национальный исследовательский Томский политехнический университет. – 2016. – С. 348–350.
42. Злыднева Т.П. Теория обработки экспериментальных данных и практика её реализации в учебном процессе университета с помощью универсального пакета Statistica. // Журнал педагогических исследований. М.: ИНФРА-М, – 2018. – Т. 3. – № 2. – С. 165–178.
43. Злыднева Т.П. Потенциал развития профессиональных компетенций студентов в процессе практико-ориентированного образования // Журнал педагогических исследований. М.: ИНФРА-М, – 2018. – Т. 3. – № 4. – С. 124–135.
44. Злыднева Т.П. Создание рабочего проекта web-приложения при обучении программированию на начальном этапе профессиональ-

ной подготовки // Журнал исследований по управлению. М.: ИНФРА-М, – 2018. – Т. 4. – № 9. – С. 10–16.
45. Злыднева Т.П. Методы диагностики уровня овладения учебно-исследовательской деятельностью // Актуальные проблемы современной науки, техники и образования: тезисы докладов 77-й международной научно-технической конференции. Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, – 2019. – Т.2. – С. 64.
46. Злыднева Т.П. Образовательные технологии как условие эффективности профессиональной подготовки // Актуальные проблемы науки и образования в современном вузе: сб. тр. IV межд. науч.-практ. конф., республика Башкортостан, г. Стерлитамак, 23 – 25 мая 2019 г. Т. II / отв. ред. А.Л. Галиев. – Стерлитамак: Стерлитамакский филиал БашГУ, – 2019. – С. 433–437.
47. Злыднева Т.П. Опыт и перспективы организации учебно-исследовательской деятельности студентов // Актуальные проблемы современной науки: состояние, тенденции развития: сб. тр. III Всероссийской научно-практической конференции. Черкесск: Изд-во ФГБОУ ВО «Северо-Кавказская государственная академия», – 2019. – С. 57–59.
48. Злыднева Т.П. Возможности современных образовательных технологий для повышения эффективности подготовки в области информатики // Современные достижения университетских научных школ: сб. докладов науч. школы-конференции. Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, – 2019, – С. 148–151.
49. Колосков П.В. Microsoft Windows XP. Популярный самоучитель. 2-е изд., перераб. и доп. – М.: Наука и техника, 2005. – 368 с.
50. Коньков К. А. Устройство и функционирование ОС Windows. Практикум к курсу «Операционные системы». – Бином. Лаборатория знаний: Москва, 2008. – 208 с.
51. Коробко И.В. Системный администратор. Справочник системного администратора по программированию Windows. – СПб.: БХВ-Петербург, 2009. – 576 с.
52. Куприянова А.В. Реестр Windows XP. Настройки, трюки и секреты: наст. книга пользователя – М.: Наука и техника, 2006. – 192 с.
53. Курячий Г.В. Операционная система UNIX: курс лекций. – Интернет-университет информационных технологий – ИНТУИТ.РУ, 2004. – 287 с.
54. Магда Ю. С. UNIX для студента; БХВ-Петербург – Москва, 2007. – 480 с.
55. Назаров, С. В. Операционные среды, системы и оболочки. Основы структурной и функциональной организации : Учеб. пособие / С. В. Назаров. – Москва : КУДИЦ-ПРЕСС, 2007. – 504 с.: ил. – ISBN 978-

5-91136-036-8. – Текст: электронный. – URL: https://znanium.com/ catalog/product/ 369379 (дата обращения: 22.09.2020). – Режим доступа: по подписке.
56. Олифер В.Г. Сетевые операционные системы. – СПб.: Питер, 2008. – 668 с.
57. Операционные системы. Основы UNIX : учебное пособие / А. Б. Вавренюк, О. К. Курышева, С. В. Кутепов, В. В. Макаров. – Москва: ИНФРА-М, 2020. – 160 с. + Доп. материалы [Электронный ресурс]. – (Высшее образование: Бакалавриат). – ISBN 978-5-16-010893-3. – Текст: электронный. – URL: https://znanium.com/catalog/ product/ 1044511 (дата обращения: 21.09.2020). – Режим доступа: по подписке.
58. Партыка, Т.Л. Операционные системы, среды и оболочки. Учебное пособие. – М.: ФОРУМ: ИНФРА-М, 2009. – 526 с.
59. Партыка, Т. Л. Операционные системы, среды и оболочки : учебное пособие / Т.Л. Партыка, И.И. Попов. – 5-е изд., перераб. и доп. – Москва : ФОРУМ : ИНФРА-М, 2020. – 560 с. – ISBN 978-5-00091-501-1. – Текст: электронный. – URL: https://znanium.com/catalog/ product/1189335 (дата обращения: 21.09.2020). – Режим доступа: по подписке.
60. Рудаков, А. В. Операционные системы и среды : учебник / Рудаков А.В. – Москва : КУРС: ИНФРА-М, 2018. – 304 с. – ISBN 978-5-906923-85-1. – Текст: электронный. – URL: https://znanium.com /catalog /product/946815 (дата обращения: 22.09.2020). – Режим доступа: по подписке
61. Таненбаум, Э. Современные операционные системы: пер. с англ., 2-е изд. – СПб.: Питер, 2006. – 1040 с.
62. Таненбаум, Э., Вудхалл А. Операционные системы. Разработка и реализация. – СПб.: Питер, 2007. – 702 с.
63. Свиридова М. Ю. Операционная система Windows XP. – Академия - Москва, 2009. – 192 с.
64. Условия гуманитаризации высшего профессионального образования: монография / Акманова С.В., Недосекина М.А., Масленникова О.Е., Яковлева Л.А., Ивлев А.В., Москвина А.А., Злыднева Т.П.; Министерство образования и науки Российской Федерации, ГОУ ВПО «Магнитогорский государственный университет». – Магнитогорск, 2008. – 462 с.

Учебное текстовое электронное издание

Злыднева Татьяна Павловна

ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ

Проблемно-информационный курс

Учебное пособие

Издание 3-е

0,94 Мб
1 электрон. опт. диск

г. Магнитогорск, 2021 год
ФГБОУ ВО «МГТУ им. Г.И. Носова»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра прикладной математики и информатики
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru