Глава 2. Управление задачами

Понятия *процесса* (process) и *потока выполнения* (thread) нам уже известны. Мы теперь знаем, в чем здесь имеется сходство, а в чем — существенное различие. Однако в данной главе при рассмотрении вопросов распределения процессорного времени мы не всегда будем разделять эти понятия. Дело в том, что по отношению к этому ресурсу — процессорному времени — оба этих понятия практически эквивалентны. Они выступают просто как некоторая работа, для выполнения которой необходимо предоставить центральный процессор. Поэтому мы будем в основном использовать термин *задача* (task), который является как бы обобщающим. Ведь каждый поток выполнения на самом деле получает статус задачи, и для него создается соответствующий дескриптор. Но мы должны помнить о различиях между дескриптором процесса и дескриптором задачи. Даже если процесс состоит из единственного потока, мы говорим о дескрипторе процесса, содержащем информацию, с помощью которой операционная система отслеживает все ресурсы, необходимые процессу для его выполнения. Один из основных модулей супервизора операционной системы — *диспетчер задач* — переводит процессы в одно из состояний в зависимости от того, доступен тот или иной ресурс или не доступен. И поскольку в мультизадачной системе любой процесс содержит хотя бы один поток, то потоку (то есть задаче) ставится в соответствие дескриптор задачи, в котором сохраняется контекст этих вычислений. Сказанное справедливо для мультипрограммных систем, поддерживающих мультизадачный режим. В мультипрограммных системах, не поддерживающих мультизадачность, контекст прерванного процесса хранится в дескрипторе этого процесса. Заметим, что повсеместно распространенные системы Windows 9x/NT/2000/XP являются и мультипрограммными, и мультизадачными. Не случайно начиная с Windows NT и Windows 95 компания Microsoft отказалась от термина «задача» и стала использовать понятия процесса и потока выполнения (треда, нити). Правда, для изложения вопросов диспетчеризации это становится неудобным, ибо здесь чаще используется обобщающее понятие.

Еще одним доводом в пользу термина «задача» при рассмотрении вопросов организации распределения процессорного времени между выполняющимися вычислениями является аналогичный выбор этой сущности разработчиками про-

цессоров. Именно для отображения этой ситуации и обеспечения дополнительными возможностями системных программистов в решении вопросов распределения процессорного времени они вводят специальные информационные структуры и аппаратную поддержку для работы с ними. Во многих современных микропроцессорах, предназначенных для построения на их основе мощных мультипрограммных и мультизадачных систем, имеются *дескрипторы задач.* Примером, подтверждающим этот тезис, являются микропроцессоры, совместимые с архитектурой ia32, то есть с 32-разрядными процессорами фирмы Intel. Основные архитектурные особенности этих микропроцессоров, специально проработанные для организации мультизадачных операционных систем, рассматриваются достаточно подробно в главе 4. Здесь мы лишь отметим тот факт, что в этих процессорах имеется специальная аппаратная поддержка организации мультизадачного (и мультипрограммного) режима. Речь идет о *сегменте состояния задачи* (Task State Segment, TSS), который предназначен, прежде всего, для сохранения контекста потока или процесса и который легко позволяет организовать и мультипрограммный, и мультизадачный режимы. Не случайно был введен термин «задача», ибо он здесь применим и по отношению к полноценному вычислительному процессу, и по отношению к легковесному процессу (потоку выполнения, треду, нити). На самом деле этот аппаратный механизм применяется гораздо реже, чем об этом думали разработчики архитектуры ia32. На практике оказалось, что для сохранения контекста потоков эффективнее использовать программные механизмы, хотя они и не обеспечивают такой же надежности, как аппаратные.

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

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

Создание задачи сопряжено с формированием соответствующей информационной структуры, а ее удаление — с расформированием. Создание и удаление задач осуществляется по соответствующим запросам от пользователей или от самих задач. Задача может породить новую задачу. При этом между задачами появляются «родственные» отношения. Порождающая задача называется «отцом», «родителем», а порожденная — «потомком». Отец может приостановить или удалить свою дочернюю задачу, тогда как потомок не может управлять отцом.

Процессор является одним из самых необходимых ресурсов для выполнения вычислений. Поэтому способы распределения времени центрального процессора между выполняющимися задачами сильно влияют и на скорость выполнения отдельных вычислений, и на общую эффективность вычислительной системы.

Основным подходом в организации того или иного метода управления процессами, обеспечивающего эффективную загрузку ресурсов или выполнение каких-либо иных целей, является организация очередей процессов и ресурсов. При распреде-

лении процессорного времени между задачами также используется механизм очередей

Решение вопросов, связанных с тем, какой задаче следует предоставить процессорное время в данный момент, возлагается на специальный модуль операционный системы, чаще всего называемый *диспетчером задач* Вопросы же подбора вычислительных процессов которые не только можно, но и целесообразно решать параллельно, возлагаются на *планировщик процессов*

Вопросы синхронизации задач и обеспечение их различными средствами передачи сообщений и данных между ними вынесены в отдельную главу, и сейчас мы их рассматривать не будем

Планирование и диспетчеризация процессов и задач

Когда говорят о диспетчеризации, то всегда в явном или неявном виде подразумевают понятие задачи (потока выполнения) Если операционная система не поддерживает механизм потоковых вычислений, то можно заменять понятие задачи понятием процесса Ко всему прочему, часто понятие задачи используется в таком контексте, что для его трактовки приходится использовать термин «процесс»

Очевидно, что на распределение ресурсов влияют конкретные потребности тех задач, которые должны выполняться параллельно Другими словами, можно столкнуться с ситуациями, когда невозможно эффективно распределять ресурсы с тем, чтобы они не простаивали Например, пусть всем выполняющимся процессам требуется некоторое устройство с последовательным доступом Но поскольку, как мы уже знаем, оно не может разделяться между параллельно выполняющимися процессами, то процессы вынуждены будут очень долго ждать своей очереди, то есть недоступность одного ресурса может привести к тому, что длительное время не будут использоваться многие другие ресурсы

Если же мы возьмем такой набор процессов, что они не будут конкурировать между собой за неразделяемые ресурсы при своем параллельном выполнении, то, скорее всего, процессы смогут выполниться быстрее (из-за отсутствия дополнительных ожиданий) да и имеющиеся в системе ресурсы, скорее всего, будут использоваться более эффективно Таким образом, возникает задача подбора такого множества процессов, которые при своем выполнении будут как можно реже конфликтовать за имеющиеся в системе ресурсы Такая задача называется *планированием вычислительных процессов*

Задача планирования процессов возникла очень давно — в первых пакетных операционных системах при планировании пакетов задач, которые должны были выполняться на компьютере и по возможности бесконфликтно и оптимально использовать его ресурсы В настоящее время актуальность этой задачи стала меньше На первый план уже очень давно вышли задачи динамического (или краткосрочного) планирования, то есть текущего наиболее эффективного распределения ресурсов, возникающего практически по каждому событию Задачи динамического

планирования стали называть *диспетчеризацией*[1]. Очевидно, что планирование процессов осуществляется гораздо реже, чем текущее распределение ресурсов между уже выполняющимися задачами. Основное различие между долгосрочным и краткосрочным планировщиками заключается в частоте их запуска, например: краткосрочный планировщик может запускаться каждые 30 или 100 мс, долгосрочный — один раз в несколько минут (или чаще; тут многое зависит от общей длительности решения заданий пользователей).

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, в случае освобождения ресурсов памяти должен быть переведен в очередь процессов, готовых к выполнению. Долгосрочный планировщик выбирает процесс из входной очереди с целью создания неоднородной мультипрограммной смеси. Это означает, что в очереди готовых к выполнению процессов должны находиться в разной пропорции как процессы, ориентированные на ввод-вывод, так и процессы, ориентированные преимущественно на активное использование центрального процессора.

Краткосрочный планировщик решает, какая из задач, находящихся в очереди готовых к выполнению, должна быть передана на исполнение. В большинстве современных операционных систем, с которыми мы сталкиваемся, долгосрочный планировщик отсутствует.

Планирование вычислительных процессов и стратегии планирования

Прежде всего, следует отметить, что при рассмотрении стратегий планирования, как правило, идет речь о краткосрочном планировании, то есть о диспетчеризации. Долгосрочное планирование, как мы уже отметили, заключается в подборе таких вычислительных процессов, которые бы меньше всего конкурировали между собой за ресурсы вычислительной системы. Иногда используется термин *стратегия обслуживания*.

Стратегия планирования определяет, какие процессы мы планируем на выполнение для того, чтобы достичь поставленной цели. Известно большое количество различных стратегий выбора процесса, которому необходимо предоставить процессор. Среди них, прежде всего, можно выбрать следующие:

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

[1] К сожалению, здесь наблюдается терминологическая несогласованность. Собственно модули супервизора, отвечающие за диспетчеризацию задач, часто называют планировщиками (scheduler). Однако фактически, говоря о тех же планировщиках памяти или о каких-нибудь других модулях, отвечающих за динамическое распределение ресурсов, имеют в виду, что эти планировщики осуществляют диспетчеризацию. Наконец, иногда диспетчеризацию называют краткосрочным планированием.

Когда говорят о стратегии обслуживания, всегда имеют в виду понятие процесса, а не понятие задачи, поскольку процесс, как мы уже знаем, может состоять из нескольких потоков выполнения (задач).

На сегодняшний день абсолютное большинство компьютеров — это персональные IBM-совместимые компьютеры, работающие на платформах Windows компании Microsoft. Это однопользовательские диалоговые мультипрограммные и мультизадачные системы. При создании операционных систем для персональных компьютеров разработчики, прежде всего, стараются обеспечить комфортную работу с системой, то есть основные усилия уходят на проработку пользовательского интерфейса. Что касается эффективности организации вычислений, то она, видимо, тоже должна оцениваться с этих позиций. Если же считать системы Windows операционными системами общего назначения, что тоже возможно, ибо эти системы повсеместно используют для решения самых разнообразных задач автоматизации, то также следует признать, что принятые в системах Windows стратегии обслуживания приводят к достаточно высокой эффективности вычислений. Некоторым даже удается использовать системы Windows NT/2000 для решения задач реального времени. Однако выбор этих операционных систем для таких задач скорее всего делается либо вследствие некомпетентности, либо из-за невысоких требований ко времени отклика и гарантиям обслуживания со стороны самих систем реального времени, которые реализуются на Windows NT/2000.

Прежде всего, система, ориентированная на однопользовательский режим, должна обеспечить хорошую реакцию системы на запросы от того приложения, с которым сейчас пользователь работает. Мало пользователей, которые могут параллельно работать с большим числом приложений. Поэтому по умолчанию для задачи, с которой пользователь непосредственно работает и которую называют *задачей переднего плана* (foreground task), система устанавливает более высокий уровень приоритета. В результате процессорное время прежде всего предоставляется текущей задаче пользователя, и он не будет испытывать лишний раз дискомфорт из-за медленной реакции системы на его запросы. Для обеспечения надлежащей работы коммуникационных процессов и для возможности выполнять системные функции приоритет задач пользователя должен быть ниже, чем у тех задач, которые реализуют операции ввода-вывода и иные управляющие функции.

Например, в Windows 2000 можно открыть окно Свойства системы, перейти на вкладку Дополнительно, щелчком на кнопке Параметры быстродействия открыть одноименное окно и с помощью переключателя в разделе Отклик приложений установить режим Оптимизировать быстродействие приложений. Это будет соответствовать выбору такой стратегии диспетчеризации задач, в соответствии с которой приоритет на получение процессорного времени будут иметь задачи пользователя, а не фоновые служебные вычисления. В предыдущей версии ОС — Windows NT 4.0 — для выбора нужной ему стратегии пользователь должен был на вкладке Быстродействие окна Свойства системы установить желаемое значение в поле Ускорение приложения переднего плана. Это ускорение можно сделать максимальным (по умолчанию), а можно его свести к нулю. Последний вариант означал бы, что все запущенные пользователем приложения будут иметь одинаковый приори-

тет. Последнее важно, если пользователь часто запускает сразу по нескольку задач, каждая из которых требует длительных вычислений, причем эти приложения часто используют операции ввода-вывода. Например, если нужно обработать несколько десятков музыкальных или графических файлов, причем каждый файл имеет большие размеры, то выполнение всей этой работы как множества параллельно исполняющихся задач будет завершено за меньшее время, если указать стратегию равенства обслуживания. Должно быть очевидным, что любой другой вариант решения этой задачи потребует больше времени. Например, последовательное выполнение задач обработки каждого файла (то есть обработка следующего файла может начинаться только по окончании обработки предыдущего) приведет к самому длительному варианту. Стратегия предоставления процессорного времени в первую очередь текущей задаче пользователя, которая установлена в системах Windows по умолчанию, приведет нас к промежуточному (по затратам времени) результату.

Очевидно, что в идеале в очереди готовых к выполнению задач должны находиться в разной пропорции как задачи, ориентированные на ввод-вывод, так и задачи, ориентированные преимущественно на работу с центральным процессором. Практически все операционные системы стараются учесть это требование, однако не всегда оно выполняется настолько удачно, что пользователь получает превосходное время реакции системы на свои запросы и при этом видит, что его ресурсоемкие приложения выполняются достаточно быстро.

Дисциплины диспетчеризации

Известно большое количество *дисциплин диспетчеризации,* то есть правил формирования очереди готовых к выполнению задач, в соответствии с которыми формируется эта очередь (список). Иногда их называют *дисциплинами обслуживания,* опуская тот факт, что речь идет о распределении процессорного времени. Одни дисциплины диспетчеризации дают наилучшие результаты для одной стратегии обслуживания, в то время как для другой стратегии они могут быть вовсе неприемлемыми. Известно большое количество дисциплин диспетчеризации. Мы же, несмотря на статус этой книги, рассмотрим далеко не все, а только те, которые признаны наиболее эффективными и до сих пор имеют применение.

Прежде всего, различают два больших класса дисциплин обслуживания: *бесприоритетные* и *приоритетные*. При бесприоритетном обслуживании выбор задач производится в некотором заранее установленном порядке без учета их относительной важности и времени обслуживания. При реализации приоритетных дисциплин обслуживания отдельным задачам предоставляется преимущественное право попасть в состояние исполнения. Перечень дисциплин обслуживания и их классификация приведены на рис. 2.1.

В концепции приоритетов имеем следующие варианты:

* приоритет, присвоенный задаче, является величиной постоянной;
* приоритет изменяется в течение времени решения задачи *(динамический приоритет)*.

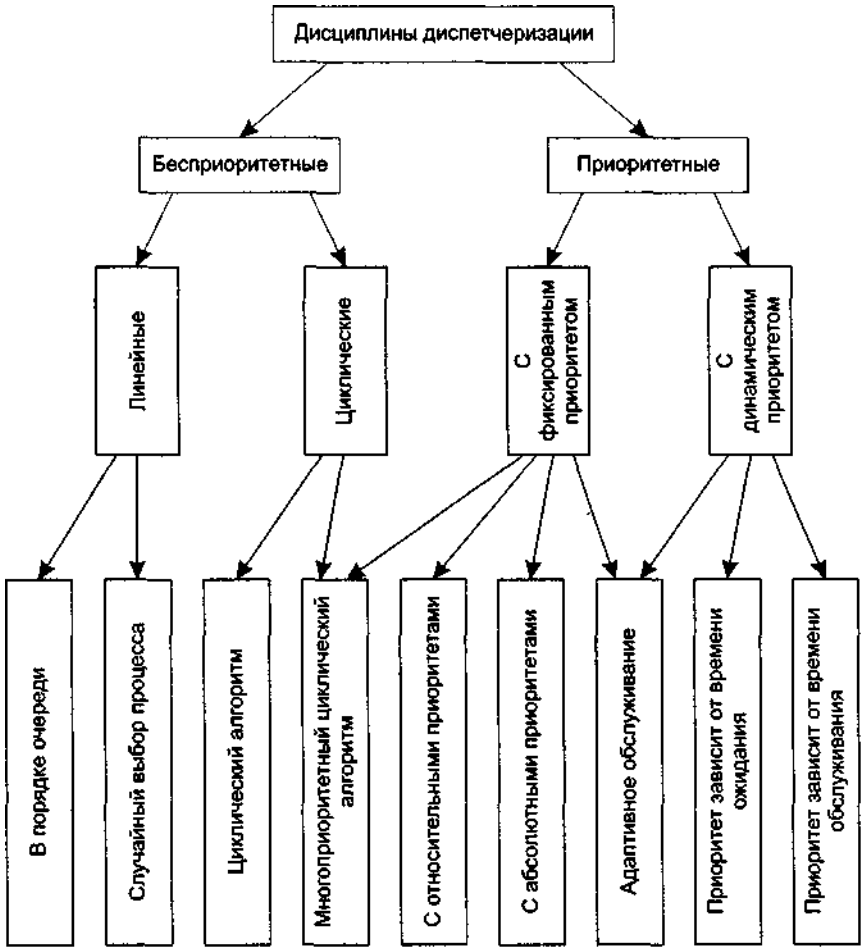

Рис. 2. 1. Дисциплины диспетчеризации

Диспетчеризация с динамическими приоритетами требует дополнительных расходов на вычисление значений приоритетов исполняющихся задач, поэтому во многих операционных системах реального времени используются методы диспетчеризации на основе *абсолютных приоритетов.* Это позволяет сократить время реакции системы на очередное событие, однако требует детального анализа всей системы для правильного присвоения соответствующих приоритетов всем исполняющимся задачам с тем, чтобы гарантировать обслуживание. Проблему гарантии обслуживания мы рассмотрим ниже.

Рассмотрим некоторые основные (наиболее часто используемые) дисциплины диспетчеризации.

Самой простой в реализации является дисциплина *FCFS* (First Come First Served — первым пришел, первым обслужен), согласно которой задачи обслуживаются «в порядке очереди», то есть в порядке их появления. Те задачи, которые были заблоки-

рованы в процессе работы (попали в какое-либо из состояний ожидания, например из-за операций ввода-вывода), после перехода в состояние готовности вновь ставятся в эту очередь готовности. При этом возможны два варианта. Первый (самый простой) — это ставить разблокированную задачу в конец очереди готовых к выполнению задач. Этот вариант применяется чаще всего. Второй вариант заключается в том, что диспетчер помещает разблокированную задачу перед теми задачами, которые еще не выполнялись. Другими словами, в этом случае образуется две очереди (рис. 2.2): одна очередь образуется из новых задач, а вторая очередь — из ранее выполнявшихся, но попавших в состояние ожидания. Такой подход позволяет реализовать стратегию обслуживания «по возможности заканчивать вычисления в порядке их появления». Эта дисциплина обслуживания не требует внешнего вмешательства в ход вычислений, при ней не происходит перераспределения процессорного времени. Про нее можно сказать, что она относится к не вытесняющим дисциплинам[1].

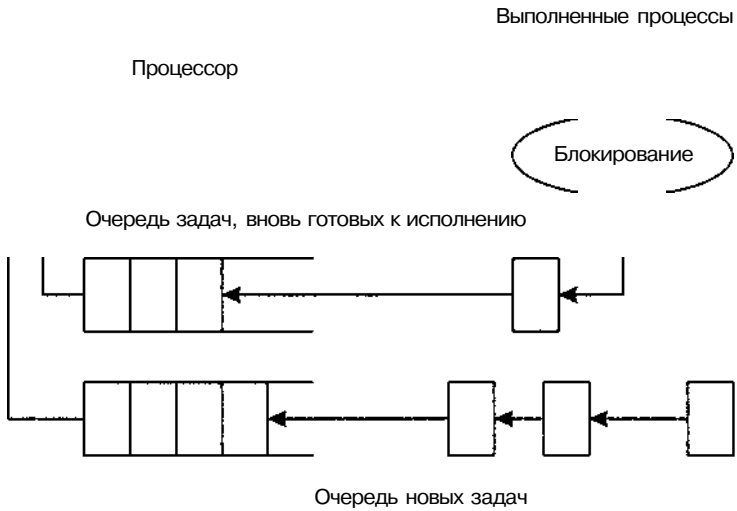

Рис. 2.2. Дисциплина диспетчеризации FCFS

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

Однако эта дисциплина приводит к тому, что при увеличении загрузки вычислительной системы растет и среднее время ожидания обслуживания, причем короткие задания (требующие небольших затрат машинного времени) вынуждены ожидать

[1] Существующие дисциплины диспетчеризации процессов могут быть разбиты на два класса: вытесняющие (preemptive) и не вытесняющие (non-preemptive). В первых пакетных операционных системах часто реализовывали параллельное выполнение заданий без принудительного перераспределения процессора между задачами. В большинстве современных ОС для мощных вычислительных систем, а также в ОС для персональных компьютеров, ориентированных на высокопроизводительное выполнение приложений (Windows 9x/NT/2000/XP, Linux, OS/2), реализованы вытесняющие дисциплины диспетчеризации (вытесняющая многозадачность).

столько же, сколько трудоемкие задания. Избежать этого недостатка позволяют дисциплины SJN и SRT. Правило FCFS применяется и в более сложных дисциплинах диспетчеризации. Например, в приоритетных дисциплинах диспетчеризации, если имеется несколько задач с одинаковым приоритетом, которые стоят в очереди готовых к выполнению задач, то попадают они в эту очередь с учетом времени.

Дисциплина обслуживания *SJN(*Shortest Job Next — следующим выполняется самое короткое задание) требует, чтобы для каждого задания была известна оценка в потребностях машинного времени. Необходимость сообщать операционной системе характеристики задач с описанием потребностей в ресурсах вычислительной системы привела к тому, что были разработаны соответствующие языковые средства. В частности, ныне уже забытый *языкJCL* (Job Control Language — язык управления заданиями) был рдним из наиболее известных. Пользователи вынуждены были указывать предполагаемое время выполнения задачи и для того, чтобы они не злоупотребляли возможностью указать заведомо меньшее время выполнения (с целью возможности получить результаты раньше других), ввели подсчет реальных потребностей. Диспетчер задач сравнивал заказанное время и время выполнения и в случае превышения указанной оценки потребности в данном ресурсе ставил данное задание не в начало, а в конец очереди. Еще в некоторых операционных системах в таких случаях использовалась система штрафов, при которой в случае превышения заказанного машинного времени оплата вычислительных ресурсов осуществлялась уже по другим расценкам.

Дисциплина обслуживания SJN предполагает, что имеется только одна очередь заданий, готовых к выполнению. Задания, которые в процессе своего исполнения были временно заблокированы (например, ожидали завершения операций ввода-вывода), вновь попадали в конец очереди готовых к выполнению наравне с вновь поступающими. Это приводило к тому, что задания, которым требовалось очень немного времени для своего завершения, вынуждены были ожидать процессор наравне с длительными работами, что не всегда хорошо.

Для устранения этого недостатка и была предложена дисциплина *SRT* (Shortest Remaining Time) — следующим будет выполняться задание, которому осталось меньше всего выполняться на процессоре.

Все эти три дисциплины обслуживания могут использоваться для пакетных режимов обработки, когда пользователю не нужно ждать реакции системы — он просто сдает свое задание и через несколько часов получает результаты вычислений. Для интерактивных же вычислений желательно прежде всего обеспечить приемлемое время реакции системы. Если же система является мультитерминальной, то помимо малого времени реакции системы на запрос пользователя желательно, чтобы она обеспечивала и равенство в обслуживании. Можно сказать, что стратегия обслуживания, согласно которой главным является равенство обслуживания при приемлемом времени обслуживания, является главной для систем разделения времени. Кстати, UNIX-системы реализуют дисциплины обслуживания, соответствующие именно этой стратегии.

Если же это однопользовательская система, но с возможностью мультипрограммной обработки, то желательно, чтобы те программы, с которыми непосредственно

работает пользователь, имели лучшее время реакции, нежели фоновые задания. При этом желательно, чтобы некоторые приложения, выполняясь без непосредственного участия пользователя (например, программа получения электронной почты, использующая модем и коммутируемые линии для передачи данных), тем не менее, гарантированно получали необходимую им долю процессорного времени.

Для решения перечисленных проблем используется дисциплина обслуживания, называемая *карусельной* (Round Robin, RR), и приоритетные методы обслуживания.

Дисциплина обслуживания RR предполагает, что каждая задача получает процессорное время порциями или, как говорят, *квантами времени* (time slice) q. После окончания кванта времени q задача снимается с процессора, и он передается следующей задаче. Снятая задача ставится в конец очереди задач, готовых к выполнению. Эту дисциплину обслуживания иллюстрирует рис. 2.3. Для оптимальной работы системы необходимо правильно выбрать закон, по которому кванты времени выделяются задачам.

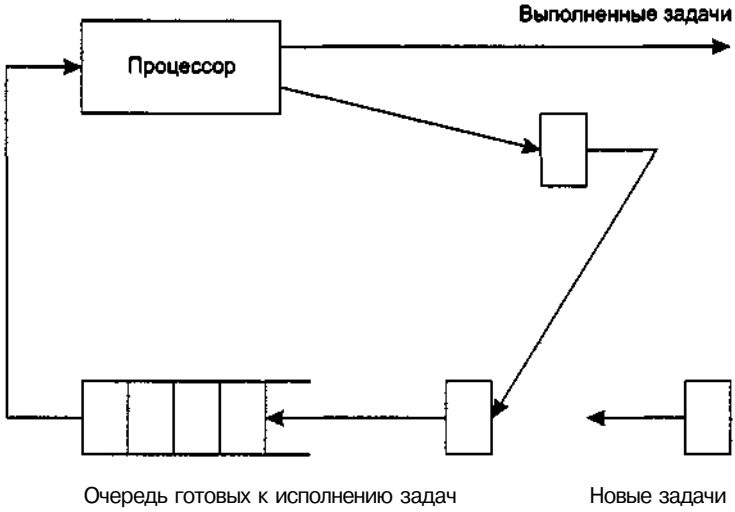

Рис. 2.3. Карусельная дисциплина диспетчеризации

Величина кванта времени q выбирается как компромисс между приемлемым временем реакции системы на запросы пользователей (с тем, чтобы их простейшие запросы не вызывали длительного ожидания) и накладными расходами на частую смену контекста задач. Очевидно, что при прерываниях операционная система вынуждена выполнять большой объем работы, связанной со сменой контекста. Она должна сохранить достаточно большой объем информации о текущем (прерываемом) процессе, поставить дескриптор снятой задачи в очередь, занести в рабочие регистры процессора соответствующие значения для той задачи, которая теперь будет выполняться (ее дескриптор расположен первым в очереди готовых к исполнению задач). Если величина q велика, то при увеличении очереди готовых к выполнению задач реакция системы станет медленной. Если же величина q мала, то относительная доля

накладных расходов на переключения контекста между исполняющимися задачами увеличится, и это ухудшит производительность системы.

В некоторых операционных системах есть возможность указывать в явном виде величину кванта времени или диапазон возможных значений, тогда система будет стараться выбирать оптимальное значение сама. Например, в операционной системе OS/2 в файле CONFIG.SYS есть возможность с помощью оператора TIMESLICE указать минимальное и максимальное значения для кванта q. Так, например, строка TIMESLICE=32,256 указывает, что минимальное значение равно 32 мс, а максимальное — 256. Если некоторая задача прервана, поскольку израсходован выделенный ей квант времени q, то следующий выделенный ей интервал будет увеличен на время, равное одному периоду таймера (около 32 мс), и так до тех пор, пока квант времени не станет равным максимальному значению, указанному в операторе TIMESLICE. Этот метод позволяет OS/2 уменьшить накладные расходы на переключение задач в том случае, если несколько задач параллельно выполняют длительные вычисления. Следует заметить, что диспетчеризация задач в этой операционной системе реализована, пожалуй, наилучшим образом с точки зрения реакции системы и эффективности использования процессорного времени.

Дисциплина карусельной диспетчеризации более всего подходит для случая, когда все задачи имеют одинаковые права на использование ресурсов центрального процессора. Однако как мы знаем, равенства в жизни гораздо меньше, чем неравенства. Одни задачи всегда нужно решать в первую очередь, тогда как остальные могут подождать. Это можно реализовать за счет того, что одной задаче мы (или диспетчер задач) присваиваем один приоритет, а другой задаче — другой. Задачи в очереди будут располагаться в соответствии с их приоритетами. Формирует очередь диспетчер задач. Процессор в первую очередь будет предоставляться задаче с самым высоким приоритетом, и только если ее потребности в процессоре удовлетворены или она попала в состояние ожидания некоторого события, диспетчер может предоставить его следующей задаче. Многие дисциплины диспетчеризации по-разному используют основную идею карусельной диспетчеризации и механизм приоритетов.

Дисциплина диспетчеризации RR — это одна из самых распространенных дисциплин. Однако бывают ситуации, когда операционная система не поддерживает в явном виде дисциплину карусельной диспетчеризации. Например, в некоторых операционных системах реального времени используется диспетчер задач, работающий по принципу абсолютных приоритетов (процессор предоставляется задаче с максимальным приоритетом, а при равенстве приоритетов он действует по принципу очередности) [7, 11]. Другими словами, причиной снятия задачи с выполнения может быть только появление задачи с более высоким приоритетом. Поэтому если нужно организовать обслуживание задач таким образом, чтобы все они получали процессорное время равномерно и равноправно, то системный оператор может сам организовать эту дисциплину. Для этого достаточно всем пользовательским задачам присвоить одинаковые приоритеты и создать одну высокоприоритетную задачу, которая не должна ничего делать, но которая, тем не менее, будет по таймеру (через указанные интервалы времени) планироваться на выполнение. Благодаря высокому приоритету этой задачи текущее приложение будет сниматься с выполнения и ставиться в конец очереди, а поскольку этой высокоприоритетной задаче

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

В своей простейшей реализации дисциплина карусельной диспетчеризации предполагает, что все задачи имеют одинаковый приоритет. Если же необходимо ввести механизм приоритетного обслуживания, то это, как правило, делается за счет *организации нескольких очередей.* Процессорное время предоставляется в первую очередь тем задачам, которые стоят в самой привилегированной очереди. Если она пустая, то диспетчер задач начинает просматривать остальные очереди. Именно по такому алгоритму действует диспетчер задач в операционных системах OS/2, Windows 9x, Windows NT/2000/XP и многих других. Отличия в основном заключаются в количестве очередей и в правилах, касающихся перемещения задач из одной очереди в другую.

Известные дисциплины диспетчеризации (мы здесь рассмотрели только основные) могут применять или не применять еще одно правило, касающееся перераспределения процессора между выполняющимися задачами.

Есть дисциплины, в которых процессор принудительно может быть отобран у текущей задачи. Такие дисциплины обслуживания называют *вытесняющими,* поскольку одна задача вытесняется другой. Другими словами, возможно принудительное перераспределение процессорного времени между выполняющимися задачами. Оно осуществляется самой операционной системой, отбирающей периодически процессор у выполняющейся задачи.

А есть дисциплины диспетчеризации, в которых ничто не может отобрать у задачи процессор, пока она сама его не освободит. Освобождение процессора в этом случае, как правило, связано с тем, что задача попадает в состояние ожидания некоторого события.

Итак, диспетчеризация без перераспределения процессорного времени, то есть *не вытесняющая* (non-preemptive multitasking), или *кооперативная, многозадачность* (cooperative multitasking), — это такой способ диспетчеризации задач, при котором активная задача выполняется до тех пор, пока она сама, что называется «по собственной инициативе», не отдаст управление диспетчеру задач для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс или поток. Дисциплины обслуживания FCFS, SJN, SRT относятся к не вытесняющим.

Диспетчеризация с перераспределением процессорного времени между задачами, то есть *вытесняющая многозадачность* (preemptive multitasking), — это такой способ, при котором решение о переключении процессора с выполнения одной задачи на выполнение другой принимается диспетчером задач, а не самой активной задачей. При вытесняющей многозадачности механизм диспетчеризации задач целиком сосредоточен в операционной системе, и программист может писать свое приложение, не заботясь о том, как оно будет выполняться параллельно с другими задачами (процессами и потоками). При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения текущей задачи, сохраняет ее контекст в дескрипторе задачи, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст. Дисциплина RR и многие другие, построенные на ее основе, относятся к вытесняющим.

При не вытесняющей многозадачности процессорное время распределено между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама должна определить момент завершения своей очередной итерации и передачи управления супервизору операционной системы с помощью соответствующего системного вызова. При этом естественно, что диспетчер задач, так же как и в случае вытесняющей мультизадачности, формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом порядка поступления задач или их приоритетов) следующую задачу на выполнение. Такой механизм создает некоторые проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что управление системой может теряться на некоторый произвольный период времени, который определяется процессом выполнения приложения (а не системой, старающейся всегда обеспечить приемлемое время реакции на запросы пользователей) [27]. Если приложение тратит слишком много времени на выполнение какой-либо работы (например, на форматирование диска), пользователь не может переключиться с этой на другую задачу (например, на текстовый или графический редактор, в то время как форматирование продолжалось бы в фоновом режиме). Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу.

Поэтому разработчики приложений для не вытесняющей операционной среды, возлагая на себя функции диспетчера задач, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Так, упомянутая выше программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста.

Например, в ныне уже забытой операционной среде Windows 3.x нативные 16-разрядные приложения этой системы разделяли между собой процессорное время именно таким образом. И именно программисты должны были обеспечивать «дружественное» отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая управление ядру системы. Крайним проявлением «недружественности» приложения является его зависание, приводящее к общему краху системы — прекращению всех вычислений. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный механизм диспетчеризации, во-первых, обеспечивает все задачи процессорным временем, во-вторых, дает возможность иметь надежные механизмы мониторинга вычислений и, в-третьих, позволяет снять зависшую задачу с выполнения.

Однако распределение функций диспетчеризации между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и достоинством, потому что дает возможность разработчику приложений самому планировать распределение процессорного времени наиболее подходящим для

данного фиксированного набора задач образом [27, 44, 46]. Так как разработчик сам определяет в программе момент времени передачи 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 каждой итерации использует их монопольно и уверена, что на протяжении этого периода никто другой их не изменит. Примером эффективного применения не вытесняющей многозадачности является сетевая операционная система Novell NetWare, в которой в значительной степени благодаря этому достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование не вытесняющей многозадачности в операционной среде Windows 3.x. К счастью, на сегодня эта операционная система уже нигде не применяется, ее с успехом заменила сначала Windows 95, а затем и Windows 98. Правда, следует заметить, что при выполнении в этих операционных системах старых 16-разрядных приложений, разработанных в свое время для операционной среды Win16 API, создается виртуальная машина, работающая по принципам не вытесняющей многозадачности.

Качество диспетчеризации и гарантии обслуживания

Одна из проблем, которая возникает при выборе подходящей дисциплины обслуживания — это *гарантия обслуживания*. Дело в том, что в некоторых дисциплинах, например в дисциплине абсолютных приоритетов, низкоприоритетные процессы получаются обделенными многими ресурсами и, прежде всего, процессорным временем. Возникает реальная дискриминация низкоприоритетных задач, в результате чего они достаточно длительное время могут не получать процессорное время. В конце концов, некоторые процессы и задачи вообще могут быть не выполнены к заданному сроку. Известны случаи, когда вследствие высокой загрузки вычислительной системы отдельные процессы вообще не выполнились, несмотря на то что прошло несколько лет (!) с момента их планирования. Поэтому вопрос гарантии обслуживания является очень актуальным.

Более жестким требованием к системе, чем просто гарантированное завершение процесса, является его гарантированное завершение к указанному моменту времени или за указанный интервал времени. Существуют различные дисциплины диспетчеризации, учитывающие жесткие временные ограничения, но не существует дисциплин, которые могли бы предоставить больше процессорного времени, чем может быть в принципе выделено.

Планирование с учетом жестких временных ограничений легко реализовать, организуя очередь готовых к выполнению задач в порядке возрастания их временных ограничений. Основным недостатком такого простого упорядочения является то, что задача (за счет других задач) может быть обслужена быстрее, чем это ей реально необходимо. Чтобы избежать этого, проще всего процессорное время выделять все-таки квантами. А после получения задачей своего кванта времени операционная система, оценив некоторое множество факторов (важных с точки зрения опти-

мизации распределения процессорного времени и гарантий обслуживания к заданному сроку), может переназначить приоритет задаче. Это позволит ей более гибко использовать механизм приоритетов и иметь механизмы гарантии обслуживания.

Гарантировать обслуживание можно, например, следующими тремя способами.

* Выделять минимальную долю процессорного времени некоторому классу процессов, если по крайней мере один из них готов к исполнению. Например, можно отводить 20 % от каждых 10 мс процессам реального времени, 40 % от каждых 2с — интерактивным процессам и 10 % от каждых 5 мин — пакетным (фоновым) процессам.
* Выделять минимальную долю процессорного времени некоторому конкретному процессу, если он готов к выполнению.
* Выделять столько процессорного времени некоторому процессу, чтобы он мог выполнить свои вычисления к сроку.

Для сравнения алгоритмов диспетчеризации обычно используются некоторые критерии.

* *Загрузка центрального процессора* (CPU utilization). В большинстве персональных систем средняя загрузка процессора не превышает 2-3 %, доходя в моменты выполнения сложных вычислений и до 100 %. В реальных системах, где компьютеры (например, серверы) выполняют очень много работы, загрузка процессора колеблется в пределах от 15-40 % (для легко загруженного процессора) до 90-100 % (для тяжело загруженного процессора).
* *Пропускная способность центрального процессора* (CPU throughput). Пропускная способность процессора может измеряться количеством процессов, которые выполняются в единицу времени.
* *Время оборота* (turnaround time). Для некоторых процессов важным критерием является полное время выполнения, то есть интервал от момента появления процесса во входной очереди до момента его завершения. Это время названо временем оборота и включает время ожидания во входной очереди, время ожидания в очереди готовых процессов, время ожидания в очередях к оборудованию, время выполнения в процессоре и время ввода-вывода.
* *Время ожидания* (waiting time). Под временем ожидания понимается суммарное время нахождения процесса в очереди готовых процессов.
* *Время отклика* (response time). Для интерактивных программ важным показателем является время отклика, или время, прошедшее от момента попадания процесса во входную очередь до момента первого обращения к терминалу.

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

Правильное планирование процессов в значительной степени влияет на производительность всей системы. Можно выделить следующие главные причины, приводящие к снижению производительности системы.

* Накладные расходы на переключение процессора. Они определяются не только переключениями контекстов задач, но и (при переключении на потоки другого приложения) перемещениями страниц виртуальной памяти, а также необходимостью обновления данных в кэше (коды и данные одной задачи, находящиеся в кэше, не нужны другой задаче и будут заменены, что приведет к дополнительным задержкам).
* Переключение на другую задачу в тот момент, когда текущая задача выполняет критическую секцию, а другие задачи активно ожидают входа в свою критическую секцию (см. главу 7). В этом случае потери будут особо велики (хотя вероятность прерывания выполнения коротких критических секций мала).

В случае мультипроцессорных систем применяются следующие методы повышения производительности системы:

* совместное планирование, при котором все потоки одного приложения (неблокированные) одновременно ставятся на выполнение процессорами и одновременно снимаются с выполнения (для сокращения переключений контекста);
* планирование, при котором находящиеся в критической секции задачи не прерываются, а активно ожидающие входа в критическую секцию задачи не ставятся на выполнение до тех пор, пока вход в секцию не освободится;
* планирование с учетом так называемых *подсказок* (hints) программы (во время ее выполнения), например, в известной своими новациями ОС Mach имелось два класса таких подсказок: во-первых, указания (разной степени категоричности) о снятии текущего процесса с процессора, во-вторых, указания о том процессе, который должен быть выбран взамен текущего.

Одним из основных методов гарантии обслуживания является использование динамических приоритетов.

Диспетчеризация задач с использованием динамических приоритетов

При выполнении программ, реализующих какие-нибудь задачи контроля и управления (что характерно, прежде всего, для систем реального времени), может случиться такая ситуация, когда одна или несколько задач не могут быть реализованы (решены) в течение длительного промежутка времени из-за возросшей нагрузки в вычислительной системе. Потери, связанные с невыполнением таких задач, могут оказаться больше, чем потери от невыполнения программ с более высоким приоритетом. При этом оказывается целесообразным временно изменить приоритет «аварийных» задач (для которых истекает отпущенное для них время обработки). После выполнения этих задач их приоритет восстанавливается. Поэтому почти в любой операционной системе реального времени (ОС РВ) имеются средства для *динамического изменения приоритета* (dynamic priority variation) задачи. Есть такие средства и во многих операционных системах, которые не относятся к классу ОС РВ.

Рассмотрим, например, как реализован механизм динамических приоритетов в операционной системе UNIX, которая, как известно, не относится к ОС РВ. Операцион-

ные системы класса UNIX относятся к мультитерминальным диалоговым системам. Основная стратегия обслуживания, применяемая в UNIX-системах, — это равенство в обслуживании и обеспечение приемлемого времени реакции системы. Реализуется эта стратегия за счет дисциплины диспетчеризации RR с несколькими очередями и механизма динамических приоритетов. Приоритет процесса вычисляется следующим образом [39]. Во-первых, в вычислении участвуют значения двух полей дескриптора процесса — p_nice и p_cpu. Первое из них назначается пользователем явно или формируется по умолчанию с помощью системы программирования. Второе поле формируется диспетчером задач (планировщиком разделения времени) и называется системной составляющей или текущим приоритетом. Другими словами, каждый процесс имеет два атрибута приоритета. С учетом этого приоритета и распределяется между исполняющимися задачами процессорное время: *текущий приоритет,* на основании которого происходит планирование, и заказанный *относительный приоритет* (называемый *nice number,* или просто *nice).*

Схема нумерации текущих приоритетов различна для различных версий UNIX. Например, более высокому значению текущего приоритета может соответствовать более низкий фактический приоритет планирования. Разделение между приоритетами режима ядра и задачи также зависит от версии. Рассмотрим частный случай, когда текущий приоритет процесса варьируется в диапазоне от 0 (низкий приоритет) до 127 (наивысший приоритет). Процессы, выполняющиеся в режиме задачи, имеют более низкий приоритет, чем в режиме ядра. Для режима задачи приоритет меняется в диапазоне 0-65, для режима ядра — 66-95 (системный диапазон). Процессы, приоритеты которых лежат в диапазоне 96-127, являются процессами с фиксированным приоритетом, не изменяемым операционной системой, и предназначены для поддержки приложений реального времени.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение *приоритета сна,* выбираемое ядром из диапазона системных приоритетов и связанное с событием, вызвавшим это состояние. Когда процесс пробуждается, ядро устанавливает значение текущего приоритета процесса равным приоритету сна. Поскольку приоритет такого процесса находится в системном диапазоне и выше, чем приоритет режима задачи, вероятность предоставления процессу вычислительных ресурсов весьма велика. Такой подход позволяет, в частности, быстро завершить системный вызов, в ходе выполнения которого могут блокироваться некоторые системные ресурсы.

После завершения системного вызова перед возвращением в режим задачи ядро восстанавливает приоритет режима задачи, сохраненный перед выполнением системного вызова. Это может привести к понижению приоритета, что, в свою очередь, вызовет переключение контекста.

Текущий приоритет процесса в режиме задачи p_priuser, как мы только что отмечали, зависит от значения относительного приоритета p_nice и степени использования вычислительных ресурсов p_cpu:

```
p_priuser = a x p_nice - b x p_cpu
```

Задача планировщика разделения времени — справедливо распределить вычислительный ресурс между конкурирующими процессами. Для принятия решения о

выборе следующего запускаемого процесса планировщику необходима информация об использовании процессора. Эта составляющая приоритета уменьшается обработчиком прерываний таймера каждый тик. Таким образом, пока процесс выполняется в режиме задачи, его текущий приоритет линейно уменьшается.

Каждую секунду ядро пересчитывает текущие приоритеты процессов, готовых к запуску (приоритеты которых меньше некоторого порогового значения; в нашем примере эта величина равна 65), последовательно увеличивая их за счет последовательного уменьшения отрицательного компонента времени использования процессора. Как результат, эти действия приводят к перемещению процессов в более приоритетные очереди и повышению вероятности их последующего выполнения.

Возможно использование следующей формулы:

```
p_cpu - p_cpu/2
```

В этом правиле проявляется недостаток нивелирования приоритетов при повышении загрузки системы. Происходит это потому, что в таком случае каждый процесс получает незначительный объем вычислительных ресурсов и, следовательно, имеет малую составляющую p_cpu, которая еще более уменьшается благодаря формуле пересчета величины p_cpu. В результате загрузка процессора перестает оказывать заметное влияние на приоритет, и низкоприоритетные процессы (то есть процессы с высоким значением nice number) практически «отлучаются» от вычислительных ресурсов системы.

В некоторых версиях UNIX для пересчета значения p_cpu используется другая формула:

```
p_cpu = p_cpu x (2 x load)/(2 x load + 1)
```

Здесь параметр load равен среднему числу процессов, находившихся в очереди на выполнение за последнюю секунду, и характеризует среднюю загрузку системы за этот период времени. Этот алгоритм позволяет частично избавиться от недостатка планирования по формуле p_cpu - p_cpu/2, поскольку при значительной загрузке системы уменьшение p_cpu при пересчете будет происходить медленнее.

Описанные алгоритмы диспетчеризации позволяют учесть интересы низкоприоритетных процессов, так как в результате длительного ожидания очереди на запуск приоритет таких процессов увеличивается, соответственно повышается и вероятность их запуска. Эти алгоритмы также обеспечивают более вероятный выбор планировщиком интерактивных процессов по отношению к сугубо вычислительным (фоновым). Такие задачи, как командный интерпретатор или редактор, большую часть времени проводят в ожидании ввода, имея, таким образом, высокий приоритет (приоритет сна). При наступлении ожидаемого события (например, пользователь осуществил ввод данных) им сразу же предоставляются вычислительные ресурсы. Фоновые процессы, потребляющие значительные ресурсы процессора, имеют высокую составляющую p_cpu и, как следствие, более низкий приоритет.

Аналогичные механизмы имеют место и в таких операционных системах, как OS/2 или Windows NT/2000/XP. Правда, алгоритмы изменения приоритета задач в этих системах иные. Например, в Windows NT/2000/XP каждый поток выполнения имеет базовый уровень приоритета, который лежит в диапазоне от двух уровней

ниже базового приоритета процесса, его породившего, до двух уровней выше этого приоритета, как показано на рис. 2.4. Базовый приоритет процесса определяет, сколь сильно могут различаться приоритеты потоков этого процесса и как они соотносятся с приоритетами потоков других процессов. Поток наследует этот базовый приоритет и может изменять его так, чтобы он стал немного больше или немного меньше. В результате получается приоритет планирования, с которым поток и начинает исполняться. В процессе исполнения потока его приоритет может отклоняться от базового.

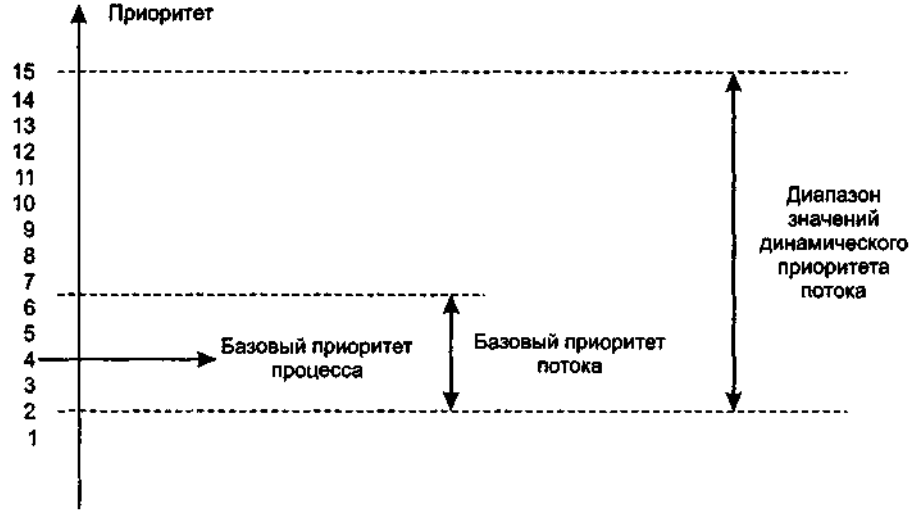

Рис. 2.4. Схема динамического изменения приоритетов в Windows NT/2000/XP

На рисунке также показан динамический приоритет потока, нижней границей которого является базовый приоритет потока, а верхняя зависит от вида работ, исполняемых потоком. Например, если поток обрабатывает текущие результаты операций ввода пользователем своих данных, диспетчер задач Windows поднимает его динамический приоритет; если же он выполняет вычисления, то диспетчер задач постепенно снижает его приоритет до базового. Снижая приоритет одной задачи и поднимая приоритет другой, подсистемы могут управлять относительной приоритетностью потоков внутри процесса.

Для определения порядка выполнения потоков диспетчер задач использует систему приоритетов, направляя на выполнение задачи с высоким приоритетом раньше задач с низким приоритетом. Система прекращает исполнение, или *вытесняет* (preempts), текущий поток, если становится готовым к выполнению другой поток с более высоким приоритетом.

Имеется группа очередей — по одной для каждого приоритета. В операционных системах Windows NT/2000/XP используется один и тот же диспетчер задач. Он поддерживает 32 уровня приоритета. Задачи делятся на два класса: реального времени и переменного приоритета. Задачи реального времени, имеющие приоритеты от 16 до 31, — это высокоприоритетные потоки, используемые программами,

критическими по времени выполнения, то есть требующими немедленного внимания системы (по терминологии Microsoft).

Диспетчер задач просматривает очереди, начиная с самой приоритетной. При этом если очередь пустая, то есть в ней нет готовых к выполнению задач с таким приоритетом, то осуществляется переход к следующей очереди. Следовательно, если есть задачи, требующие процессор немедленно, они будут обслужены в первую очередь. Для собственно системных модулей, функционирующих в статусе задачи, зарезервирована очередь с номером 0.

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

В операционной системе OS/2 схема динамической приоритетной диспетчеризации несколько иная, хоть и похожа[1]. В OS/2 также имеется четыре класса задач. И для каждого класса задач имеется своя группа приоритетов с интервалом значений от 0 до 31. Итого, 128 различных уровней и, соответственно, 128 возможных очередей готовых к выполнению задач (потоков).

Задачи, имеющие самые высокие значения приоритета, называются *критически ми по времени* (time critical). В этот класс входят задачи, которые мы в обиходе называем *задачами реального времени,* то есть для них должен быть обязательно предоставлен определенный минимум процессорного времени. Наиболее часто встречающимися задачами этого класса являются задачи коммуникаций (например, задача управления последовательным портом, на который приходят биты по коммутируемой линии с подключенным модемом, или задачи управления сетевым оборудованием). Если такие задачи не получат управление в нужный момент времени, то сеанс связи может прерваться.

[1] Как известно, одно время компания Microsoft принимала активное участие в разработке OS/2 совместно с IBM. Поэтому после прекращения совместных работ над этой операционной системой и начале нового проекта многие решения из OS/2 были унаследованы в варианте OS/2 ver. 3.0, названной впоследствии Windows NT.

Следующий класс задач имеет название *приоритетного.* Поскольку к этому классу относят задачи, которые выполняют по отношению к остальным задачам функции сервера (о модели клиент-сервер, по которой строятся современные операционные системы с микроядерной архитектурой, см. главы 9 и 10), то его еще иногда называют *серверным.* Приоритет таких задач должен быть выше, поскольку это позволяет гарантировать, что запрос на некоторую функцию со стороны обычных задач выполнится сразу, а не будет дожидаться, пока до него дойдет очередь на фоне других пользовательских приложений.

Большинство задач относят к обычному классу, его еще называют *регулярным* (regular), или *стандартным.* По умолчанию система программирования порождает задачу, относящуюся именно к этому классу.

Наконец, существует еще класс фоновых задач, называемый в OS/2 *остаточным.* Программы этого класса получают процессорное время только тогда, когда нет задач из других классов, требующих процессор. В качестве примера такой задачи можно привести программу обновления индексного файла, используемого при поиске файлов, или программу проверки электронной почты.

Внутри каждого из вышеописанных классов задачи, имеющие одинаковый уровень приоритета, выполняются в соответствии с дисциплиной RR. Переход от одного потока к другому происходит либо по окончании отпущенного ему кванта времени, либо по системному прерыванию, передающему управление задаче с более высоким приоритетом (таким образом система вытесняет задачи с более низким приоритетом для выполнения задач с более высоким приоритетом и может обеспечить быструю реакцию на важные события).

OS/2 самостоятельно изменяет приоритет выполняющихся программ независимо от уровня, установленного самим приложением. Этот механизм называется *повышением приоритета* (priority boost). Операционная система изменяет приоритет задачи в трех случаях [26].

* Повышение приоритета активной задачи (foreground boost). Приоритет задачи автоматически повышается, когда она становится активной. Это снижает время реакции активного приложения на действия пользователя по сравнению с фоновыми программами.
* Повышение приоритета ввода-вывода (Input/Output boost). По завершении операции ввода-вывода задача получает самый высокий уровень приоритета ее класса. Таким образом обеспечивается завершение всех незаконченных операций ввода-вывода.
* Повышение приоритета «забытой» задачи (starvation boost). Если задача не получает управление в течение достаточно долгого времени (этот промежуток времени задает оператор MAXWAIT в файле CONFIG.SYS[1]), диспетчер задач OS/2 временно присваивает ей уровень приоритета, не превышающий критический. В результате переключение на такую «забытую» программу происходит быстрее. После выполнения приложения в течение одного кванта времени его при-

[1] Строка MAXWAIT = 1 означает, что приоритет задачи при переключении на нее будет поднят до максимального не позже чем через одну секунду.

оритет вновь снижается до остаточного. В сильно загруженных системах этот механизм позволяет программам с остаточным приоритетом работать хотя бы в краткие интервалы времени. В противном случае они вообще никогда бы не получили управление.

Если нет необходимости использовать метод динамического изменения приоритета, то с помощью оператора PRIOPITY - ABSOLUTE в файле CONFIG.SYS можно ввести дисциплину абсолютных приоритетов; по умолчанию оператор PRIOPITY имеет значение DYNAMIC.

Контрольные вопросы и задачи

1. Перечислите и поясните основные функции операционных систем, которые связаны с управлением задачами.
2. В чем заключается основное различие между планированием процессов и диспетчеризацией задач?
3. Что такое стратегия обслуживания? Перечислите известные вам стратегии обслуживания.
4. Какие дисциплины диспетчеризации задач вы знаете? Поясните их основные идеи, перечислите достоинства и недостатки.
5. Расскажите, какие дисциплины диспетчеризации следует отнести к вытесняющим, а какие — к не вытесняющим.
6. Как можно реализовать механизм разделения времени, если диспетчер задач работает только по принципу предоставления процессорного времени задаче с максимальным приоритетом?
7. Что такое «гарантия обслуживания»? Как ее можно реализовать?
8. Опишите механизм динамической диспетчеризации, реализованный в UNIX-системах.
9. Сравните механизмы диспетчеризации задач в операционных системах Windows NT и OS/2. В чем они похожи друг на друга и в чем заключаются основные различия?