УДК 519.7

И.С. Гусев

АЛГОРИТМ ПЛАНИРОВАНИЯ, ОСНОВАННЫЙ НА СПРАВЕДЛИВОМ РАСПРЕДЕЛЕНИИ ПРОЦЕССОРНОГО ВРЕМЕНИ

Предлагаются классификация операционных систем и задач; алгоритм планирования, основанный на равномерном распределении процессорного времени.

На современном этапе развития операционных систем все чаще возникает необходимость работы в режиме реального времени, а также выполнения распределенных вычислений. Обычно производители операционных систем проводят разделение на системы реального времени и системы распределенного времени. Потребитель может приобрести либо систему разделения времени (например, для произведения математических вычислений), либо реального времени для некоторых производственных процессов.

Однако далеко не всегда подходят вышеперечисленные крайние случаи. В большинстве автоматизированных систем редко требуется задействовать всю мощь процессора, в то время как хотелось бы выполнять и задачи, связанные не только с реальным временем. Системы же, как правило, рассчитаны для задач чисто реального времени или чисто распределения процессорного времени. Зачастую для систем реального времени не требуется все процессорное время, а только его определенный процент, но вовремя, т.е. через гарантированные промежутки времени. Рассмотрим, например, следующий случай. В многопользовательской системе, где пользователи проводят интенсивные математические вычисления, в том числе запускают дополнительные сервисы, должен работать сервис, отвечающий за считывание данных с некоторого датчика, и для удовлетворительной работы этого сервиса требуется 10 % процессорного времени. Однако этот сервис должен получать свое время регулярно, иначе он будет пропускать данные, так как буфер датчика невелик и требует частого, но кратковременного считывания. Конечно, эту ситуацию можно решить простым планированием по наивысшему приоритету, но если таких сервисов в системе несколько, то метод по наивысшему приоритету не помогает.

Классификация задач и операционных систем

Для лучшего понимания и конструирования алгоритмов планирования необходимо определить основные классы задач и операционных систем. В табл. 1 приводятся предлагаемые классы задач в современных операционных системах. Очевидно, что задачи всех классов не могут существовать в операционной системе одновременно. В табл. 2 приводятся классы операционных систем и набор задач, которые могут в них исполняться одновременно.

Таблица 1

Классы задач

Класс задач	Описание задач
Задачи жесткого реального времени (HRTT)	Получают все процессорное время, если готовы к выполнению. Все остальные задачи в системе не получают процессорного времени, пока выполняются эти задачи. Они выполняются, только когда критические задачи реального времени заблокированы при ожидании ответа внешнего устройства или завершения сервисной процедуры, такой как чтение или запись файлов. Таких задач не может быть больше, чем процессоров в системе.
Задачи мягкого реального времени (SRTT)	Не требуют всего процессорного времени. Им достаточно только некоторого процента, однако они требуют регулярного получения очередных квантов процессорного времени. Сумма времени, требуемого такими задачами, не должна превышать 100 %.
Системные задачи (SysT)	Сами по себе эти задачи ничего полезного не делают, только обслуживают запросы других задач. Как правило, они используются для обслуживания внешних устройств общего назначения, например дисков, сетевых карт и т.д. Для обслуживания специализированных внешних устройств используются задачи реального времени.
Задачи разделения времени (STT)	Являются пользовательскими задачами. Время для этих задач распределяется в соответствии с некоторым критерием справедливости.
Фоновые задачи (BgT)	Получают процессорное время, только если нет готовых задач других классов. Для этих задач совершенно не гарантируется время их завершения или то, что им вообще будет выделено процессорное время. Как правило, в системах, особенно персональных, всегда есть время, когда процессор простаивает. Особенно это время велико, если в программе предполагается диалог с пользователем.

Таблица 2

Классы операционных систем

Класс системы	Классы задач
Системы экстремального реального времени (ERTS)	1. Задачи жесткого реального времени (HRTT). 2. Системные задачи (SysT). 3. Фоновые задачи (BgT).
Системы реального времени (RTS)	1. Задачи мягкого реального времени (SRTT). 2. Системные задачи (SysT). 3. Задачи разделения времени (STT). 4. Фоновые задачи (BgT).
Системы разделения времени (ShS)	1. Системные задачи (SysT). 2. Задачи разделения времени (STT). 3. Фоновые задачи (BgT).

Алгоритмы планирования

Невозможно создать алгоритм планирования, который бы мог подходить под любой случай. Однако создать алгоритм планирования можно достаточно эффективно для определенного класса операционных систем. Чем больше классов задач поддерживает операционная система, тем сложнее будет алгоритм планирования. Если система не поддерживает задач реального времени или число задач в системе невелико, можно использовать классические алгоритмы планирования. Если задач в системе много (а это стало возможным из-за значительного увеличения памяти) или необходимо гарантированное и справедливое распределение процессорного времени, то классические подходы невозможно использовать. Однако если использовать их комбинации с некоторыми доработками, то они могут стать вполне пригодными.

Требования оптимальности к алгоритму планирования для классов задач и классов операционных систем

Прежде всего, определимся, каким должен быть оптимальный алгоритм планирования процессорного времени и какие у него самые важные характеристики. Процессорное время распределяется квантами – единицами времени, в течение которого работает задача и по истечении которого процессор может быть отдан другой задаче или оставлен текущей. Синхронизация или момент завершения квантов происходит по таймеру, точнее по прерыванию таймера. Таким образом, можно сказать, что квант – это время t между двумя тиками системного таймера. Однако для процессора прерывание таймера – это простое прерывание от внешнего устройства, а он просто выполняет поток инструкций и реагирует на внешние события, например тот же таймер. Для работы алгоритма планирования также необходимо процессорное время, как правило, он запускается на тике таймера, т.е. в начале кванта, и собственно работает в течение времени qt (где q<1), отбирая время у этого кванта, пока не определит, какой задаче отдать то, что осталось от кванта. В некоторых алгоритмах с целью уменьшения q вводится дополнительный алгоритм отложенного планирования, который вызывается не на каждый квант, а через N квантов, и соответственно он планирует задачи на последующие N квантов. Естественно, что для этого алгоритма тоже требуется время, определим его как pt, где, как правило, p>>1. На рис. 1 приводится диаграмма работы процессора.

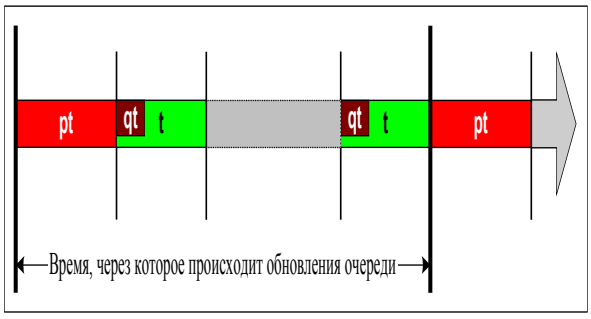

Рис. 1. Диаграмма планирования процессорного времени

В идеальном алгоритме планирования параметры p и q должны быть равны нулю. Очевидно, что построить идеальный алгоритм планирования невозможно, но тем не менее можно построить оптимальный алгоритм планирования в зависимости от требования к этому алгоритму. У различных задач эти требования различны, и невозможно построить требования оптимальности для всего множества задач. Оптимальные требования можно построить только для определенного класса задач, а затем попытаться перенести эти требования на все классы задач определенного класса операционной системы и тем самым определить оптимальные требования к алгоритму планирования конкретного класса ОС. В общем случае полученный алгоритм будет лишь приближенно оптимальным, и только в редких случаях он будет оптимальным для одного класса задач операционной системы.

Основные требования к алгоритмам планирования:

1. Использовать минимальное время для работы. Этот критерий является общим для всех классов задач, но он всегда вторичен по отношению к базовому критерию, который определяется функциональностью.
2. Обеспечить справедливость с точностью до определенного времени (время может быть равно нулю). Этот критерий является базовым для задач разделения времени.
3. Предоставить максимально быструю реакцию на внешние события. Этот критерий является базовым для задач реального времени.
4. Обеспечить гарантированное стабильное расстояние между выделяемыми квантами. Этот критерий является базовым для задач мягкого реального времени.

Классические алгоритмы планирования: преимущества и недостатки

Рассмотрим классические алгоритмы [1, 2], их достоинства и недостатки.

Круговорот (round robin). Первая задача из очереди готовых задач получает квант времени длиной t, а затем отправляется снова в конец очереди, если только она себя не заблокирует. Возможны варианты с учетом приоритета. Для каждой задачи выделяется сразу несколько квантов подряд в зависимости от приоритета. Недостаток – группирование квантов, что приводит к неравномерному выполнению высокоприоритетных задач. Для обеспечения плавного выполнения необходимо уменьшать параметр q, что приводит к дополнительным накладным расходам.

Планирование по наивысшему приоритету. Несколько высокоприоритетных задач могут захватить процессорное время на неопределенно длительное время. А поскольку в системе могут находиться сервисы, которые могут вообще не завершаться, то низкоприоритетные задачи могут вообще не получить процессорное время.

Очереди с обратной связью. В этих алгоритмах используются очереди, работающие так же, как в алгоритме круговорота, и следовательно, имеют те же самые недостатки. Имеет место выигрыш при использовании множества коротких заданий. В действительности много задач имеют неограниченное время выполнения.

Многоуровневое планирование. Обычно метод многоуровневого планирования реализуется как система, состоящая из трех уровней: диспетчера, краткосрочного планировщика и долгосрочного планировщика. Диспетчер выбирает следующую подлежащую выполнению задачу и работает в течение времени *qt*. Краткосрочный планировщик вызывается, чтобы вставить го-товый процесс в очередь. Долгосрочный планировщик вызывается редко (например, через каждые 100 квантов), и занимается он трудоемкими операциями, такими как пересчет приоритетов и подготовка списка задач для запуска до последующего своего запуска в течение времени *p*. В частности, такой метод используется в ОС Unix [3]. В данном методе не происходит равномерного распределения времени на коротких промежутках времени, так как во время между вызовами долгосрочного планировщика приоритеты могут измениться и, следовательно, содержать неактуальную информацию. Более того, для пересчета приоритетов требуется линейная трудоемкость и, следовательно, время *p* зависит от числа задач, что совершенно неприемлемо для систем реального времени.

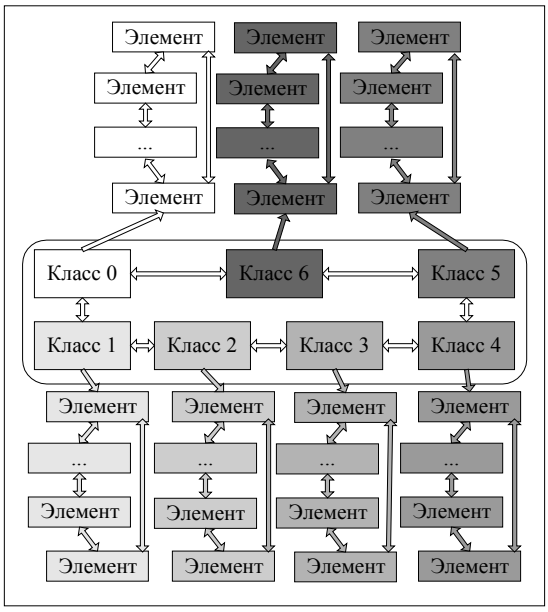

Рис. 2. Система классов приоритетов

Справедливый алгоритм планирования

Предлагаемый алгоритм обеспечивает справедливое распределение квантов процессорного времени между пользователем и процессами. Под справедливым распределением времени понимается то, что процессорное время распределяется между пользователями и процессами только в зависимости от приоритетов и никоим образом не зависит от числа процессов, запущенных пользователем, или от числа потоков, которые имеет процесс.

В предлагаемом методе используются статические приоритеты от 0 до 6. Такое малое количество приоритетов объясняется использованием логарифмической зависимости между приоритетом и выделяемым процессорным временем. Разница между приоритетами в 1 соответствует двукратной разнице выделяемого времени. Например, процесс с приоритетом 3 будет получать в 2 раза больше квантов, чем процесс с приоритетом 4, или процесс с приоритетом 0 будет получать в 128 раз больше квантов, чем процесс с приоритетом 6.

Для описания структуры данных справедливого алгоритма вводится понятие системы классов. Элементами системы классов могут быть пользователи, процессы, потоки, а также системы классов для организации многоуровневой иерархии. Всем элементам системы классов назначается приоритет от 0 до 6, при этом все эти элементы делятся на 7 классов по приоритетам. Во время выполнения из класса 0 выбираются все задачи, из класса 1 выбирается половина задач, из класса 2 выбирается четверть задач и т.д. Элементы в классах объединяются в двухсвязные циклические списки, сами классы также объединяются в циклический двухсвязный список, образуя тем самым систему классов. Более детально система классов изображена на рис. 2.

Планировщик задач выбирает задачи из пула задач, готовых к выполнению. Пул состоит из иерархического дерева систем классов (рис. 3).

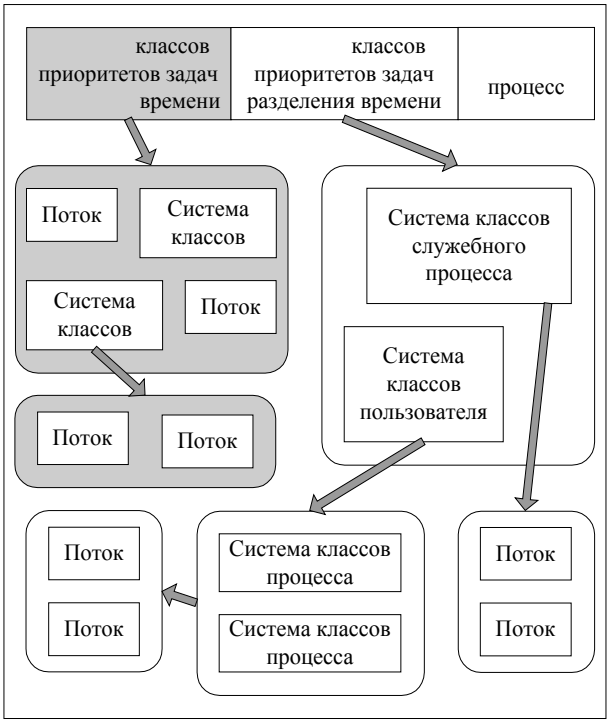

Рис. 3. Пул задач, готовых к выполнению

На самом верху находится система классов пользователя. Элементами этой системы являются системы классов процессов. Элементами системы классов процессов являются системы классов потоков. Иногда некоторые уровни иерархии могут отсутствовать. Это зависит от типа потоков, а также от установленных флагов для планировщика задач. Например, системные потоки (потоки драйверов и модулей ядра) не имеют пользователя.

Планировщик задач действует следующим образом:

1. Если система классов пользователей пуста, то возвращает пустую задачу. Текущей системой классов назначает систему классов пользователей.

2. Пока элемент не поток, выбирает элемент из текущей системы классов.

3. Возвращает выбранный поток.

Данный алгоритм имеет следующие достоинства:

– обеспечивает гарантированное время, через которое будет получен очередной квант процессорного времени;

– происходит равномерное распределение квантов процессорного времени в диапазоне квантов, равном сумме всех выделяемых квантов за полный обход (цикл) алгоритма планирования. Это позволяет создать иллюзию гладкого выполнения программ пользователя. На рис. 4 приводится диаграмма распределения квантов процессора между тремя задачами, причем одна из задач получает в два раза больше квантов, чем две остальные;

– позволяет избежать длительного игнорирования пользователя в системах мягкого реального времени;

– имеет трудоемкость алгоритма $O(1)$ и, следовательно, не зависит от числа задач. Также отсутствует необходимость в отложенном планировании, т.е. p равно нулю;

– алгоритм может быть легко модифицирован для использования в системе реального времени. Для этого достаточно в иерархию добавить еще одну систему классов, в которую помещать задачи реального времени с соответствующими приоритетами, а одним элементом этой системы поместить как раз систему классов пользователей. Таким образом, все задачи реального времени будут получать равномерно и вовремя свои кванты независимо от числа пользователей системы и запущенных ими процессов.

Рассмотрим пример работы алгоритмов планирования. Допустим, что в системе готовыми являются три задачи, но при этом необходимо, чтобы одна из них работала быстрее других в два раза. На рис. 4 приведены диаграммы работы распределения квантов.

Рассмотрим другой пример работы системы мягкого реального времени. Допустим, что в системе готовыми являются две задачи: одна обычная, а

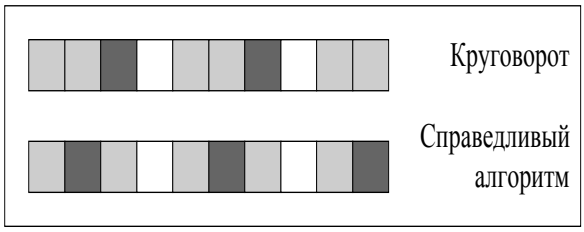

Рис. 4. Распределение квантов задач разделения времени

другая мягкого реального времени. Для работы задачи мягкого реального времени необходимо N квантов, в то время как результат можно получать в течение количества квантов, намного большего, чем N. На рис. 5 приведена диаграмма распределения квантов для этого примера.

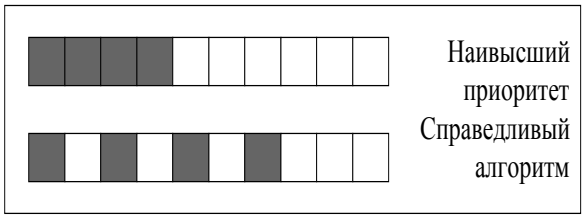

Рис. 5. Распределение квантов задач разделения времени

Комбинирование справедливого алгоритма планирования с классическими

Комбинирование возможно на разных уровнях. Как базовый можно использовать некоторый классический алгоритм, например алгоритм планирования по наивысшему приоритету, а все задачи с одинаковым приоритетом для базового алгоритма можно помещать в систему классов предложенного алгоритма. Другой способ – для системных задач использовать алгоритм планирования по наивысшему приоритету, а для задач разделения времени – предложенный алгоритм.

ЛИТЕРАТУРА

1. *Кейслер С.* Проектирование операционных систем для малых ЭВМ: Пер. с англ. М.: Мир, 1986. 680 с.
2. *Цикритзис Д., Бернстайн Ф.* Операционные системы. М.: Мир, 1977. 333 с.
3. *Maurice J. Bach.* The design of the UNIX operating system. Prentice-Hall, 1986.

Статья представлена кафедрой теоретических основ информатики факультета информатики Томского государственного университета, поступила в научную редакцию номера 3 декабря 2001 г.