Введение в планирование. Категории алгоритмов планирования. Задачи алгоритмов планирования.

Планирование заданий. Введение в планирование

Введение в планирование. Категории алгоритмов планирования. Задачи алгоритмов планирования. Планирование в системах пакетной обработки данных. Планирование в интерактивных системах. Планирование в системах реального времени.

Программное управление специальными регистрами маски (маскирование сигналов прерывания) позволяет реализовать различные дисциплины обслуживания:

·с *относительными приоритетами*, то есть обслуживание не прерывается даже при наличии запросов с более высокими приоритетами. После окончания обслуживания данного запроса обслуживается запрос с наивысшим приоритетом. Для организации такой дисциплины необходимо в программе обслуживания данного запроса наложить маски на все остальные сигналы прерывания или просто отключить систему прерываний;

· с *абсолютными приоритетами*, то есть всегда обслуживается прерывание с наивысшим приоритетом. Для реализации этого режима необходимо на время обработки прерывания замаскировать все запросы с более низким приоритетом. При этом возможно многоуровневое прерывание, то есть прерывание программ обработки прерываний. Число уровней прерывания в этом режиме изменяется и зависит от приоритета запроса;

· по *принципу стека*, или, как иногда говорят, по дисциплине *LCFS* (last come - a прерывания по нарушению питания;

Управление ходом выполнения задач со стороны ОС заключается в организации реакций на прерывания, в организации обмена информацией (данными и программами), предоставлении необходимых ресурсов, в динамике выполнения задачи и в организации сервиса. Причины прерываний определяет ОС (модуль, который называют супервизором прерываний), она же и выполняет действия, необходимые при данном прерывании и в данной ситуации. Поэтому в состав любой ОС реального времени прежде всего входят программы управления системой прерываний, контроля состояний задач и событий, синхронизации задач, средства распределения памяти и управления ею, а уже потом средства организации данных (с помощью файловых систем и т. д.). Следует, однако, заметить, что современная ОС реального времени должна вносить в аппаратно-программный комплекс нечто большее, нежели просто обеспечение быстрой реакции на прерывания.

Рассмотрим кратко основные виды ресурсов вычислительной системы и способы их разделения. Прежде всего, одним из важнейших ресурсов является сам процессор, точнее — процессорное время. Процессорное время делится попеременно (параллельно).

Вторым видом ресурсов вычислительной системы можно считать память. Оперативная память может быть разделена и одновременным способом (то есть в памяти одновременно может располагаться несколько процессов или, по крайней мере, текущие фрагменты, участвующие в вычислениях), и попеременно (в разные моменты оперативная память может предоставляться для разных вычислительных процессов). Память — очень интересный вид ресурса. Дело в том, что в каждый конкретный момент времени процессор при выполнении вычислений обращается к очень ограниченному числу ячеек оперативной памяти. С этой точки зрения желательно память разделять для возможно большего числа параллельно исполняемых процессов. С другой стороны, как правило, чем больше оперативной памяти может быть выделено для

конкретного текущего процесса, тем лучше будут условия для его выполнения. Поэтому проблема эффективного разделения оперативной памяти между параллельно выполняемыми вычислительными процессами является одной из самых актуальных

Когда говорят о внешней памяти (например, память на магнитных дисках), то собственно память и доступ к ней считаются разными видами ресурса. Каждый из этих ресурсов может предоставляться независимо от другого. Но для полной работы с внешней памятью необходимо иметь оба этих ресурса. Собственно внешняя память может разделяться одновременно, а доступ к ней — попеременно.

Если говорить о внешних устройствах, то они, как правило, могут разделяться параллельно, если используются механизмы прямого доступа. Если же устройство работает с последовательным доступом, то оно не может считаться разделяемым ресурсом. Простыми и наглядными примерами внешних устройств, которые не могут быть разделяемыми, являются принтер и накопитель на магнитной ленте. Действительно, если допустить, что принтер можно разделять между двумя процессами, которые смогут его использовать попеременно, то результаты печати, скорее всего, не смогут быть использованы — фрагменты выведенного текста могут перемешаться таким образом, что в них невозможно будет разобраться. Аналогично обстоит дело и с накопителем на магнитной ленте. Если один процесс начнет что-то читать или писать, а второй при этом запросит перемотку ленты на ее начало, то оба вычислительных процесса не смогут выполнить свои вычисления.

Уровни планирования

Раннее рассматривая эволюцию компьютерных систем, мы рассматривали *планирование* в вычислительных системах: *планировании* заданий и *планировании* использования процессора. *Планирование* заданий появилось в *пакетных системах* после того, как для хранения сформированных *пакетов заданий* начали использоваться магнитные диски. Магнитные диски, являясь устройствами прямого доступа, позволяют загружать задания в *компьютер* в произвольном порядке, а не только в том, в котором они были записаны на *диск*. Изменяя порядок загрузки заданий в вычислительную систему, можно повысить эффективность ее использования. Процедуру выбора очередного задания для загрузки в машину, т. е. для порождения соответствующего процесса, мы и назвали ***планированием*** заданий. *Планирование* использования процессора впервые возникает в мультипрограммных вычислительных системах, где в состоянии готовность могут одновременно находиться несколько процессов. Именно для процедуры выбора из них одного процесса, который получит *процессор* в свое распоряжение, т. е. будет переведен в состояние *исполнение*, мы использовали это *словосочетание*. Теперь, познакомившись с концепцией процессов в вычислительных системах, оба вида *планирования* мы будем рассматривать как различные ***уровни планирования процессов***.

Планирование заданий используется в качестве ***долгосрочного планирования процессов***. Оно отвечает за порождение новых процессов в системе, определяя ее ***степень мультипрограммирования***, т. е. количество процессов, одновременно находящихся в ней. Если *степень мультипрограммирования* системы поддерживается постоянной, т. е. среднее количество процессов в компьютере не меняется, то новые процессы могут появляться только после завершения ранее загруженных. Поэтому *долгосрочное планирование* осуществляется достаточно редко, между появлением новых процессов могут проходить минуты и даже десятки минут. Решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно длительного времени. Отсюда и название этого *уровня планирования – долгосрочное*. В некоторых операционных системах *долгосрочное планирование* сведено к минимуму или отсутствует вовсе. Так, например, во многих интерактивных системах разделения времени *порождение процесса* происходит сразу после появления соответствующего запроса. Поддержание разумной *степени мультипрограммирования* осуществляется за счет ограничения количества пользователей, которые могут работать в системе, и особенностей человеческой психологии. Если между нажатием на клавишу и появлением символа на экране проходит 20–30 секунд, то многие пользователи предпочтут прекратить работу и продолжить ее, когда система будет менее загружена.

Планирование использования процессора применяется в качестве ***краткосрочного планирования процессов***. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени. Поэтому *краткосрочное планирование* осуществляется, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т. е. в течение короткого промежутка времени, чем и обусловлено название этого *уровня планирования – краткосрочное*.

В некоторых вычислительных системах бывает выгодно для повышения производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила название swapping, что можно перевести на русский язык как "перекачка", хотя в специальной литературе оно употребляется без перевода – свопинг. Когда и какой из процессов нужно перекачать на диск и вернуть обратно, решается дополнительным промежуточным *уровнем планирования процессов* – ***среднесрочным*** .

Критерии планирования и требования к алгоритмам

Для каждого *уровня планирования процессов* можно предложить много различных алгоритмов. Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя *планирование*. К числу таких целей можно отнести следующие:

- Справедливость – гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не начинал выполняться.
- Эффективность – постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов, готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90%.
- Сокращение полного времени выполнения (***turnaround time***) – обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.
- Сокращение времени ожидания (***waiting time***) – сократить время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.
- Сокращение времени отклика (***response time***) – минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей *планирования* желательно также, чтобы алгоритмы обладали следующими свойствами.

- Были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма *планирования* не должно приводить, к примеру, к извлечению квадратного корня из 4 за сотые доли секунды при одном запуске и за несколько суток – при втором запуске.
- Были связаны с минимальными накладными расходами. Если на каждые 100 миллисекунд, выделенные процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, применять не стоит.
- Равномерно загружали ресурсы вычислительной системы, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы.
- Обладали масштабируемостью, т. е. не сразу теряли работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

Многие из приведенных выше целей и свойств являются противоречивыми. Улучшая работу алгоритма с точки зрения одного *критерия*, мы ухудшаем ее с точки зрения другого. Приспосабливая *алгоритм* под один *класс* задач, мы тем самым дискриминируем задачи другого класса.

Параметры планирования. Категории алгоритмов планирования

В различных условиях окружающей среды требуются разные алгоритмы планирования. Это обусловлено тем, что различные сферы приложений предназначены для решения разных задач. При этом стоит различать три среды:

1) пакетную;

2) интерактивную;

3) реального времени;

В пакетных системах не бывает пользователей, терпеливо ожидающих за своими терминалами быстрого ответа на свой короткий вопрос. Поэтому для них зачастую приемлемы неприоритетные алгоритмы или приоритетные алгоритмы с длительными периодами для каждого процесса. Такой подход сокращает количество переключений между процессами, повышая при этом производительность работы системы. Пакетные алгоритмы носят весьма общий характер и часто находят применение также в других ситуациях.

В среде с пользователями, работающими в интерактивном режиме, приобретает важность приоритетность, сдерживающая отдельный процесс от захвата центрального процесса, лишающего при этом доступа к службе всех других процессов. Для предупреждения такого поведения необходимо использование приоритетного алгоритма. Под эту категорию попадают и серверы.

В системах, ограниченных условиями реального времени, приоритетность иногда не требуется, поскольку процессы знают, что они могут запускаться только на непродолжительные периоды времени, и зачастую выполняют свою работу довольно быстро, а затем блокируются. В отличии от интерактивных систем в системах реального времени запускаются лишь те программы, которые, предназначены для содействия определенной прикладной задаче. Интерактивные системы имеют универсальный характер и могут запускать произвольные программы, которые не выполняют совместную задачу или даже вредят друг другу.

Для осуществления поставленных целей разумные алгоритмы *планирования* должны опираться на какие-либо характеристики процессов в системе, заданий в очереди на загрузку, состояния самой вычислительной системы, иными словами, на ***параметры*** *планирования* . В этом разделе мы опишем ряд таких *параметров*, не претендуя на полноту изложения.

Все *параметры планирования* можно разбить на две большие группы: статические *параметры* и динамические *параметры*. Статические *параметры* не изменяются в ходе функционирования вычислительной системы, динамические же, напротив, подвержены постоянным изменениям.

К статическим *параметрам* вычислительной системы можно отнести предельные значения ее ресурсов (размер оперативной памяти, максимальное количество памяти на диске для осуществления свопинга, количество подключенных устройств ввода-вывода и т. п.). Динамические *параметры* системы описывают количество свободных ресурсов на данный момент.

К статическим *параметрам* процессов относятся характеристики, как правило присущие заданиям уже на этапе загрузки.

- Каким пользователем запущен процесс или сформировано задание.
- Насколько важной является поставленная задача, т. е. каков *приоритет* ее выполнения.
- Сколько процессорного времени запрошено пользователем для решения задачи.
- Каково соотношение процессорного времени и времени, необходимого для осуществления операций ввода-вывода.
- Какие ресурсы вычислительной системы (оперативная память, устройства ввода-вывода, специальные библиотеки и системные программы и т. д.) и в каком количестве необходимы заданию.

Алгоритмы *долгосрочного планирования* используют в своей работе статические и динамические *параметры* вычислительной системы и статические *параметры* процессов (динамические *параметры* процессов на этапе загрузки заданий еще не известны). Алгоритмы *краткосрочного* и *среднесрочного планирования* дополнительно учитывают и динамические характеристики процессов. Для *среднесрочного планирования* в качестве таких характеристик может использоваться следующая *информация*:

- сколько времени прошло с момента выгрузки процесса на диск или его загрузки в оперативную память;
- сколько оперативной памяти занимает процесс;
- сколько процессорного времени уже предоставлено процессу.

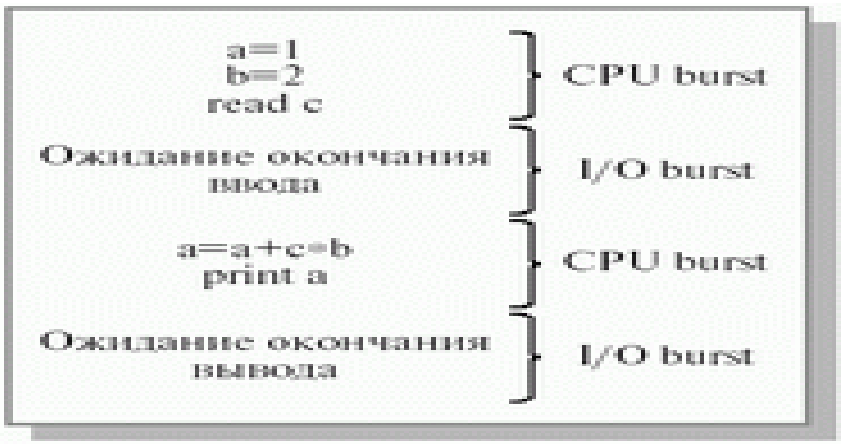

Рис.1. Фрагмент деятельности процесса с выделением промежутков непрерывного использования процессора и ожидания ввода-вывода

Для *краткосрочного планирования* нам понадобится ввести еще два динамических *параметра*. *Деятельность* любого процесса можно представить как последовательность циклов использования процессора и ожидания завершения операций ввода-вывода. Промежуток времени непрерывного использования процессора носит название ***CPU burst***, а промежуток времени непрерывного ожидания ввода-вывода – ***I/O burst*** . На рисунке 1. показан фрагмент деятельности некоторого процесса на псевдоязыке программирования с выделением указанных промежутков. Для краткости мы будем использовать термины *CPU burst* и *I/O burst* без перевода. Значения продолжительности последних и очередных *CPU burst* и *I/O burst* являются важными динамическими *параметрами* процесса.

Вытесняющее и невытесняющее планирование

Процесс *планирования* осуществляется частью операционной системы, называемой планировщиком. *Планировщик* может принимать решения о выборе для исполнения нового процесса из числа находящихся в состоянии готовность в следующих четырех случаях.

1. Когда процесс переводится из состояния исполнение в состояние закончил исполнение.
2. Когда процесс переводится из состояния исполнение в состояние ожидание.
3. Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).
4. Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция ввода-вывода или произошло другое событие). Подробно процедура такого перевода рассматривалась в лекции 2 (раздел "Переключение контекста"), где мы показали, почему при этом возникает возможность смены процесса, находящегося в состоянии исполнение.

В случаях 1 и 2 процесс, находившийся в состоянии *исполнение*, не может дальше исполняться, и *операционная система* вынуждена осуществлять планирование выбирая новый процесс для выполнения. В случаях 3 и 4 *планирование* может как проводиться, так и не проводиться, *планировщик* не вынужден обязательно принимать решение о выборе процесса для выполнения, процесс, находившийся в состоянии *исполнение* может просто продолжить свою работу. Если в

операционной системе планирование осуществляется только в вынужденных ситуациях, говорят, что имеет *место невытесняющее (nonpreemptive) планирование*. Если *планировщик* принимает и вынужденные, и невынужденные решения, говорят о *вытесняющем (preemptive) планировании*. Термин *"вытесняющее планирование"* возник потому, что исполняющийся процесс помимо своей воли может быть вытеснен из состояния *исполнение* другим процессом.

Невытесняющее планирование используется, например, в MS Windows 3.1 и ОС Apple Macintosh. При таком режиме *планирования* процесс занимает столько процессорного времени, сколько ему необходимо. При этом переключение процессов возникает только при желании самого исполняющегося процесса передать управление (для ожидания завершения операции ввода-вывода или по окончании работы). Этот метод *планирования* относительно просто реализуем и достаточно эффективен, так как позволяет выделить большую часть процессорного времени для работы самих процессов и до минимума сократить *затраты* на переключение контекста. Однако при *невытесняющем планировании* возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, из-за ошибки в программе) зацикливается и не может передать управление другому процессу. В такой ситуации спасает только перезагрузка всей вычислительной системы.

Вытесняющее планирование обычно используется в системах разделения времени. В этом режиме *планирования* процесс может быть приостановлен в любой момент исполнения. Операционная система устанавливает специальный таймер для генерации *сигнала прерывания* по истечении некоторого интервала времени – *кванта*. После прерывания процессор передается в распоряжение следующего процесса. Временные прерывания помогают гарантировать приемлемое время отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают "зависание" компьютерной системы из-за зацикливания какой-либо программы.

Задачи алгоритма планирования

Чтобы создать алгоритм планирования, нужно иметь некое представление о том, с чем должен справиться толковый алгоритм. Некоторые задачи зависят от среды, но есть и такие задачи, которые желательно выполнить в любом случае.

Вот некоторые задачи алгоритма планирования, которых следует придерживаться при различных обстоятельствах:

- все системы (равнодоступность, принуждение к определенной политике, баланс);

- пакетные системы (производительность, оборотное время, использование центрального процессора);

- интерактивные системы (время отклика, пропорциональность);

- системы реального времени (соблюдение предельных сроков, предсказуемость).

Литература:

- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Современные операционные системы, Э. Таненбаум