Алгоритмы планирования необходимы в том случае если у нас один (или несколько, но в данном случае мы будем рассматривать один) процессор и нам необходимо выбрать, какой из процессов ожидающих выполнения выполнить следующим. Часть ОС, ответственная за этот выбор - планировщик. Алгоритмы его работы будут разными на разных системах (из-за разных задач).\\

В среде с приоритетным интерактивным режимом особую важность играет приоритетность, удерживающая некоторые процессы от исполнения процессором. Для предупреждения возникновения процессов, вечно удерживающих критическую секцию (работающих вечно), используются приоритетные алгоритмы. Такими алгоритмами пользуются например сервера, обслуживающие сразу несколько торопливых клиентов единовременно.\\

Есть несколько задач, которые должны быть решены во всех системах, а не только в интерактивной:\\

• равнодоступность — предоставление каждому процессу справедливой доли времени центрального процессора;\\

• принуждение к определенной политике — наблюдение за выполнением установленной политики;\\

• баланс — поддержка загруженности всех составных частей системы.\\

А так же присуще только интерактивной системе:\\

• время отклика — быстрый ответ на запросы;\\

• пропорциональность — оправдание пользовательских надежд.\\

Производительность системы зачастую считается по трем показателям:\\

производительность, оборотное время и степень задействования центрального процессора. \\

В интерактивных системах важнее всего свести к минимуму время отклика (от ввода команды до результата). Так же определенные процессы должны иметь приоритет выше чем у остальных.\\\\

Один из самых частых алгоритмов в интерактивных системах - алгоритм циклического планирования:\\

- каждому процессу назначается интервал времени на выполнение - квант.\\

- Если за время кванта процесс не выполнился, то ресурсы ЦП у него изымаются и передаётся другому. \\

- У планировщика должен быть список готовых к выполнению процессов.\\

В этом алгоритме единственный варьируемый вариант - продолжительность одного кванта времени. Если он слишком мал, то поскольку время на переключения контекста величина постоянная, то у нас доля времени, потраченного процессом впустую (на переключение карт памяти, перезагрузку кеша и сброс на диск) будет неоправданно велика. А если вант слишком велик, то при большом количестве процессов с большими квантами времени на выполнения приведут к тому, что последний из них будет ждать возможности поработать слишком долго (это особенно обидно с учётом того, что ему возможно даже не понадобится весь квант на работу и он завершится раньше). Так что улучшения характеристик этого алгоритма зависит от поиска "золотой середины" времени кванта.\\\\

В циклическом планировании видно, что все процессы воспринимаются как равнозначные, что тоже не совсем отражает действительность. Так что был придуман алгоритм приоритетного планирования:

- каждому процессу присваивается определенный приоритет.\\

- планировщик понижает приоритет процесса с каждым его прерыванием.\\

- планировщик выбирает из ждущих процесс с наивысшим приоритетом.\\

Второй вариант:\\

- каждому процессу выделен свой определённый максимальный квант времени выполнения.\\

- когда квант выполнения исчерпан, следующим работает процесс с наивысшим приоритетом.\\

В UNIX - системах есть команда nice, позволяющая пользователю добровольно снизить приоритет своего процесса, чтобы угодить другим пользователям, но ею никто никогда не пользуется.\\\\

Есть небольшие сложности и с ограничением по скорости работы устройств ввода-вывода. Процессы, завязанные на это, большую часть времени проводят именно в ожидании ввода-вывода. Как только подобному процессу понадобится ЦП, он должен быть предоставлен немедленно, чтобы процесс смог приступить к обработке следующего запроса параллельно с другим процессом, занятым вычислениями. Если заставить процесс, ограниченный скоростью работы устройств ввода-вывода, долго ждать предоставления центрального процессора, это будет означать, что он занимает оперативную память неоправданно долго.\\

Зачастую два предыдущих алгоритма используют в связке, порождая так называемые "классы приоритетности". Внутри каждого класса используется циклический алгоритм, а между классами - приоритетный, т.е. сначала работают все из "старшего" класса, затем из среднего и т. д.\\\\

Поскольку предоставление первоочередного запуска самым коротким заданиям приводит к минимизации среднего времени отклика для пакетных систем, было бы неплохо воспользоваться этим же принципом и для интерактивных процессов.\\

Этим преимуществом пользуется метод выбора следующим самого короткого процесса:\\

Для этого планировщик должен оценить предыдущее поведения процесса с самым коротким вычисленным временем выполнения.

Технология вычисления следующей оценки $T\_n$ в серии путем расчета взвешенной

суммы текущего измеренного значения и предыдущих вычислений иногда называется

распределением по срокам давности. Она применяется во многих ситуациях, где на

основе предыдущих значений нужно выдавать какие-нибудь предсказания. Распределение по срокам давности особенно просто реализуется при a = $\frac{1}{2}$.Все, что при этом нужно, — добавить новое значение к текущей оценке и разделить сумму на 2\\\\

Следующий алгоритм - Гарантированное планирование:\\

По факту алгоритм даёт нам некоторые обещания и завязан на исполнении этих обещаний. Например при $n$ пользователей в системе, ОС может обещать что нам достанется $\frac{1}{n}$ мощности ЦП.\\

Дабы выполнить это обещание, система должна вести учёт, сколько процессорного времени ушло на текущий процесс с момента его создания. После некоторым манипуляций с исходными данными планировщик приходит к некому соотношению израсходованного и отпущенного времени ЦП для текущего процесса (0.5 - использована половина всего отпущенного времени. 1.5 - использовано в полтора раза больше положенного). В общих чертах алгоритм выглядит следующим образом:\\

- планировщик получает соотношение и оценивает его.\\

- если оно больше или равно 1, то планировщик забирает ресурсы у текущего процесса.\\

- планировщик передаёт ресурсы следящему процессу с самым низким соотношением.\\

- процесс будет работать пока его соотношение не превысит соотношение его ближайшего конкурента.\\

- выбирается следующий процесс.\\\\

Алгоритм лотерейного планирования:\\

- Процессам раздаются "лотерейные билеты" на использование системных ресурсов.\\

- при необходимости принять решение планировщик случайно "тянет" лотерейный билет и его обладателю выдаются необходимые ресурсы. \\

- более важным процессам, чтобы повысить их шансы на выигрыш, могут выдаваться дополнительные билеты.\\

- розыгрыш проводится $n$ раз в секунду и на $b$ миллисекунд работы ЦП.\\

Лотерейное планирование выгодно тем, что очень быстро приспосабливается к изменениям (например, к появлению нового процесса). Взаимодействующие процесс могут помогать друг другу и "делится билетами". Т.е. клиент отдаёт билеты серверу, чтобы тот имел больше шансов побыстрее "пробиться к процессору". После работы сервер отдаёт все билеты клиенту (ибо они ему без клиентов не нужны). Лотерейное планирование может быть использовано для решения проблем, с которыми трудно справиться другими методами. В качестве одного из примеров можно привести видеосервер.\\

Следующий алгоритм - справедливое планирование:\\

При использовании предыдущих алгоритмов 1 процесс у пользователя А имел 10\% ресурсов процессора если у пользователя Б было 9 процессов и 90\% ресурсов. \\

Чтобы избежать подобной ситуации, некоторые системы перед планированием работы процесса берут в расчет, кто является его владельцем. В этой модели каждому пользователю распределяется некоторая доля процессорного времени и планировщик выбирает процессы, соблюдая это распределение. Таким образом, если каждому из двух пользователей было обещано по 50 % процессорного времени, то они его получат, независимо от количества имеющихся у них процессов.

В качестве примера рассмотрим систему с двумя пользователями, каждому из которых обещано 50\% процессорного времени. У первого пользователя четыре процесса, A, B, C и D, а у второго пользователя только один процесс — E. Если используется циклическое планирование, то возможная последовательность планируемых процессов, соответствующая всем ограничениям, будет иметь следующий вид:\\

AEBECEDEAEBECEDE...

Но если первому пользователю предоставлено вдвое большее время, чем второму, то мы можем получить следующую последовательность:

ABECDEABECDE…\\

Разумеется, существует масса других возможностей, используемых в зависимости от применяемых понятий справедливости.\\

Управление свободной памятью

Если память распределяется в динамическом режиме, то управлять этим должна операционная система. В общих чертах, существуют два способа отслеживания использования памяти: битовые матрицы и списки свободного пространства.

При использовании битовых матриц память делится на единичные блоки размером от

нескольких слов до нескольких килобайт. С каждым единичным блоком соотносится

один бит в битовой матрице, который содержит 0, если единичный блок свободен, и 1,

если он занят (или наоборот).Важным вопросом для разработчика является размер единичного блока памяти. Чем

меньше блок, тем больше битовая матрица. Но даже с таким небольшим единичным

блоком памяти, размер которого равен 4 байта, для 32 бит памяти понадобится 1 бит

матрицы.

Битовая матрица предоставляет довольно простой способ отслеживания слов памяти

в фиксированном объеме памяти, поскольку ее размер зависит только от размера памяти и размера единичного блока памяти. Основная проблема заключается в том, что

при решении поместить в память процесс, занимающий k единичных блоков, диспетчер

памяти должен искать в битовой матрице непрерывную последовательность нулевых

битов. Поиск в битовой матрице последовательности заданной длины — довольно медленная операция.

Другим способом отслеживания памяти является ведение связанных списков распределенных и свободных сегментов памяти, где сегмент либо содержит процесс,

либо является пустым пространством между двумя процессами.Каждая запись в списке хранит обозначение, содержит сегмент «дыру» — hole (H) или

процесс — process (P), адрес, с которого сегмент начинается, его длину и указатель на следующую запись.Поскольку запись в таблице процессов, относящаяся к завершающемуся процессу, будет, как правило, указывать на запись в списке именно для этого процесса, возможно,

удобнее будет вести не односвязный список, как показано на рис. 3.6, в, а двусвязный

список. Такая структура облегчит поиск предыдущей записи и определение возможности объединения.Когда процессы и пустые пространства содержатся в списке отсортированными по

адресам, то для выделения памяти создаваемому процессу (или существующему

процессу, загружаемому в результате свопинга с диска) могут быть использованы

несколько алгоритмов. Предположим, что диспетчер памяти знает, сколько памяти

нужно выделить. Простейший алгоритм называется «первое подходящее».Незначительной вариацией алгоритма «первое подходящее» является алгоритм «следующее подходящее». Он работает так же, как и «первое подходящее», за исключением того, что отслеживает свое местоположение, как только находит подходящее пустое

пространство.

Другой хорошо известный и широко используемый алгоритм — «наиболее подходящее». При нем поиск ведется по всему списку, от начала до конца, и выбирается наименьшее соответствующее пустое пространство. Вместо того чтобы разбивать большое

пустое пространство, которое может пригодиться чуть позже, алгоритм «наиболее

подходящее» пытается подыскать пустое пространство, близкое по размеру к необходимому, чтобы наилучшим образом соответствовать запросу и имеющимся пустым

пространствам.

При попытке обойти проблему разбиения практически точно подходящих пространств

памяти на память, отводимую под процесс, и небольшие пустые пространства можно

прийти к идее алгоритма «наименее подходящее», то есть к неизменному выбору самого большого подходящего пустого пространства, чтобы вновь образующееся пустое

пространство было достаточно большим для дальнейшего использования. Моделирование показало, что применение алгоритма «наименее подходящее» также далеко не

самая лучшая идея.Работа всех четырех алгоритмов может быть ускорена за счет ведения отдельных

списков для процессов и пустых пространств. При этом все усилия этих алгоритмов

сосредоточиваются на просмотре списков пустых пространств, а не списков процессов.

Неизбежной ценой за это ускорение распределения памяти становится дополнительное

усложнение и замедление процедуры ее освобождения, поскольку освободившийся сегмент должен быть удален из списка процессов и внесен в список пустых пространств.. Когда при работе алгоритма «наиболее подходящее»

поиск пустых пространств ведется от самых маленьких до самых больших, то при обнаружении подходящего пространства становится понятно, что найденное пространство

является наименьшим, в котором может быть выполнено задание, следовательно, оно

и есть наиболее подходящее.

Еще один алгоритм распределения памяти называется «быстро искомое подходящее».

Его использование предусматривает ведение отдельных списков для некоторых наиболее востребованных искомых размеров. К примеру, у него может быть таблица из

n записей, в которой первая запись является указателем на вершину списка пустых

пространств размером 4 Кбайт, вторая — указателем на список пустых пространств размером 8 Кбайт, третья — указателем на список пустых пространств размером 12 Кбайт и т. д. Пустые пространства размером, скажем, 21 Кбайт могут быть помещены либо в список пустых пространств размером 20 Кбайт, либо в специальный список пустых пространств с нечетным размером.

При использовании алгоритма «быстро искомое подходящее» поиск пустого пространства требуемого размера выполняется исключительно быстро, но в нем имеется

недостаток, присущий всем системам, сортирующим пустые пространства по размеру,

а именно когда процесс завершается или выгружается процедурой свопинга, слишком

много времени тратится на то, чтобы определить, можно ли высвобождаемое пространство объединить с соседними. Если не проводить объединение, то память очень быстро

окажется разбитой на большое количество небольших по размеру пустых фрагментов,

в которых не смогут поместиться процессы.