17.12.20

Первый подход: создать в системе условия, при которых возникновение тупиков в принципе невозможно .

Второй подход: ?

Третий подход: устранения условия неперераспределяемости.

Алгоритм Банкира - обход тупиков: тупики в принципе возможны, но менеджер ресурсов выполняет определенные действия, в результате которых тупик не допускается.

Безопасное состояние - сущ последовательность процессов, в которой каждый из процессов может гарантированно завершиться.

Необходимо выполнить n! проверок.

в текущий момент времени можно считать, что кол-во процессов в системе фиксированное.

Для повторно используемых ресурсов характерно постоянное кол-во единиц этих ресурсов в системе. ВЫход из строя ресурсов является аварийной ситуацией в системе.

К повторно используемым ресурсам относятся семафоры. В системе сущ системная таблица, в которой отслеживаются все наборы семафоров. То же самое - о программных каналах, о сокетах и тд.

Из-за огромных затрат на выполнение алга банкира, он имеет теоретическое значение.

12:37 Haberman

P1, ..., PN - N процессов

кол-во ресурсов = a

// заявка

в-р b = (b1, ..., bN) - макс запрос ресурсов каждого процесса

Каждый процесс делает заявку claim, где указывает свои максимальные потребности в кажом из ресурсов - такая заявка может быть реализована в виде вектора

// текущее распределение

c (c1, ..., cN) - показывает общее выдление рес для процессов (сколько единиц ресурса выделено каждому процессу)

=> Речь идёт только об одном ресурсе.

говорят, что состояние ресуросов реализуемо (ресурсы могут быть выделены) если выполняются:

    1. для любого k bk <=a

    2. c <= b

    3. \sum\_i^N c\_i <= a

    Число свободных ресурсов в текущий момент времени t определяется, как

    r(t) = a - \sum\_i^N c\_i

* 4. r(t) >= 0

Если реализуемое состояние гарантирует отсутствие тупиков, оно называется безопасным, иначе - небезопасным относительно тупиков.

Реализуемое состояние, которое описывается (a, b, c) является безопасным, если существует такая последовательность процессов S, что для каждого Py принадлежащего S, ( by <= [r(t) + \sum{ k<=i } c\_ik(t)] ) (\*)

Условие (\*) показывает, что запрос i-того процесса на j-й ресурс P\_ij не должен превышать сумму свободных ресурсов и сумму ресурсов, которые будут освобождены процессами, которые предшествуют i-ому процессу в последовательности процессов.

Для этой концепции безопасного состояния важным является то, что походу такого состояния можно выполнять действия без тупиков, даже в такой ситуации, когда процессы запрашивают дополнительные ресурсы. Естественно, у сис-мы должно быть необходимое кол-во свободных единиц ресурсов, чтобы она могла удовлетворить доп-ные запросы процессов.

Важно: существует ли вообще возможность проанализировать состояние системы? Необходимо иметь информацию о кол-ве процессов, заявках, кол-ве ресурса (свободных единиц ресурса).

Если процесс может получить нужную ему единицу ресурса, он можзет завершиться и вернуть системе занимаемые единицы ресурсы. Суммарно ресурсы смогут удовлетворить потребности следующего процесса в последовательности.

Если сост ресурсов безопасне и частичная последовательность S процессов удв условию (\*), то она может быть продлена. Если все процессы до k-того могут завершить, то можно продлить на k+1-й процесс и тд.

Это предварительное необходимое условие: анализ.

31:00

Алгоритм определения безопасного состояния

// пустая последовательность удов (\*)

1. S = 0

// пробуем расширить S так, чтобы после добавления процесса k, S продолжила удов (\*)

2. Добавление в S процесса Pk

может быть неудачным:

    a. все процессы уже в S -- сост безопасное.

    b. ни одно добавление не удов (\*)  -- сост небезопасное.

S - частичная послед-ть

S\* -- дополение к   S

T -- набор процессов для добавления в  S

R -- число свободных ед. ресурсов

S = 0

T = S\*

// пока есть кандидаты

while (T <> 0)

{

    Candidate = T[1];

    T = T - {T[1]};      // ушел из кандидатов

    if (B[Candidate ] - C[Candidate ]) <= R + \sum\_{i \in S} c[i]

  {

      S = S U {Candidate } ;

      T = S\*

  }

}

SAFE = (S\* = 0); // количество кандидатов стало равным 0, определяем состояние системы как безопасное

Порядок формирования запросов может отличаться от последоавтельности процессов - в этом случае, определение безопасного состояни может быть выполнено для каджого нового запроса.**Каждый новый запрос любого процесса проверяется на безопасное состояние системы.**

Очевидно, что это строгая последовательность дейсвий и данный алгоритм может быть упрощен благодаря след утверждениям:

    1)  Если безопасное состояние модифицировано путем назначения доп ресурсов процессу P\_k и некоторая посл-ть S, которая мб частичной, содержит процесс P\_k  и удв условию (\*), то это новое состояние так же безопасно.

48:30

Пример работы алгоритма

// N = число процессов  = 5

// a = макс число ресурсов в сис-ме = 10

// b = вектор заявок процессов на ресурс

// c = вектор текущего распределения

N=5, a=10

b = (5, 8, 7, 6, 9)

с = (2, 3, 1, 1, 3) // в сумме 10 == a

// вып усл \sum c\_i <= a

// Данное состояние безопасное ? (оно реализуемое?)

// (1) и (3) - условия см выше

2 + 3 + 1 + 1 + 3 = 10 -> (1) - (3) удововлетворяются -> реализуемое

/// условия

    1. для любого k bk <=a

    2. c <= b

    3. \sum\_i^N c\_i <= a

    4. r(t) = a - \sum\_i^N c\_i >= 0

///

S = 0;

T = S\*: {1, 2, 3, 4, 5}

candidate = 1;

    T = {2, 3, 4, 5}

    b1 - c1 = 5 - 2 = 3 <= 0 -> нет

candidate = 2;

    T = {3, 4, 5}

    b2 - c2 = 8 - 3 = 5 <= 0 -> нет

candidate = 3;

    T = {4, 5}

    b3 - c3 = 7 - 1 = 6 <= 0 -> нет

candidate = 4;

    T = {5}

    b4 - c4 = 6 - 1 = 5 <= 0 -> нет

candidate = 5;

    T = 0

    b5 - c5 = 9 - 3 = 6 <= 0 -> нет

------------------------------------------

T = 0, S\*  =/=  0

-> состояние явл небезопасным

пример 2:

----------------------------------------------

N=5, a=10

b = (5, 8, 7, 6, 9)

с = (1, 1, 1, 1, 2) // в сумме 6 == a - 4

// 4 свобод ед ресурса

2 + 3 + 1 + 1 + 3 = 6 -> 10 - 6 = 4 (свобод единицы ресурса)

S = 0;

T = S\*: {1, 2, 3, 4, 5}

candidate = 1;

    T = {2, 3, 4, 5}

    b1 - c1 = 5 - 1 = 4 <= 4 + 0 -> да

// только 4 свобод ед ресурса, процесс 1 может завершиться и вернет сис-ме единицу ресурса

candidate = 2;

    T = {3, 4, 5}

    b2 - c2 = 8 - 1 = 7 <= 4 + 1 -> нет

// даже если 1й процесс вернет сис-ме ед ресурса, в сумме это не удв потребность второго процесса

candidate = 3;

    T = {4, 5}

    b3 - c3 = 7 - 1 = 6 <= 4 + 1 -> нет

// второй процесс не может завершиться, только первый - но в сумме этого недостаточно, чтобы удв потребность третьего процесса

candidate = 4;

    T = {5}

    b4 - c4 = 6 - 1 = 5 <= 4 + 1 -> да

// ему понадобится 5 ед ресурса, условие вып - 4й процесс может успешно завершиться: он вернет системе единицу ресурса - будет 6 свободных ед

 опять проходим последовательность, начиная со второго процесса

b2 - c2 = 8 - 1 = 7 <= 4 + 1  +1 -> нет

// не сможет успешно завершиться

b3 - c3 = 7 - 1 = 6 <= 4 + 1 + 1 -> да

// может успешно завершиться

S = {1, 4} U {3} = {1, 4, 3}

T = {2, 5}

candidate = 2

b2 - c2 = 8 - 1 = 7 <= 4 + 1  +1  +1-> да

// завершится и вернет 1

T = {5}

candidate = 5

b5 - c5 = 9 - 2 = 7 <= 4 +1 +1 +1 + 1 -> да

----------------------------------------------------

S = {1, 4, 3, 2, 5}

T = 0

S\* = 0 -> SAFE

Для одного типа ресурса задача разрешима

Действительно, в теории тупиков вводится понятие двухроматического (?) направленного графа. Систему можно описать таким графом. Почему? 2 подмножества вершин графа: процессы и ресурсы. Эти подмножества не пересекаются.

Возникновение тупика в системе возможно только в результате запроса

Матрица тек распредления.

B = {r, p}

b\_ij = (ri, pj) (?)

Матрица запросов

B = {p, r}

a\_ij - запрос iого процесса на jый ресурс

a\_ij = (pi, rj)

Вектор свободных ресурсов

F = [ f\_j \_ -- кол-во свободных единиц j-го ресурса]

Можем получить в результате сл. выражение:

Количество распределенных единиц j-го ресурса + количество свободных единиц этого ресурса в сумме должны дать общее количество единиц данного ресурса системе.

Улучшенный алгоритм основан на анализе запросов путем сравнения векторов.

В самом деле: пусть имеется два вектора: цэ и дэ. Тогда отношение .... Значит каждый элемент вектора сэ меньше или равен соответствующему элементу вектора дэ.  В результате, если итый процесс запршивает джитый ресурс и при этом строка запросов итого процесса меньше либо равна вектору ЭФ, то такой запрос может быть удовлетворен и может быть выполнены соответствующие сокрашения. То есть, такой процесс может получить необходимый ему ресур и соответственно продолжит ьсвое выполнение в результате освободит ьзангимаемый им ресурсы. Т. Е, дуги выходящие из вершины этого процесса мб сокращены.

Друзя мои я прилгашаю вас 24 числа на лекцию где мы закончим с вами материал о тупиках бла бла бла...