Продолжение про обход тупиков.

Пример (небезопасная ситуация):

Процессы	Текущее распр-е	Заявки	Свободных единиц ресурса
P1	5	12	
P2	2	4	
P3	4	8	
			1

Для этой ситуации нельзя найти посл-ть процессов, которые могут гарантированно завершиться;

Такое сост-е системы наз. небезопасным относительно тупика, т.к. нет посл-ти процессов, которые могут гарантированно завершиться;

Но такое состояние системы не обязательно приведёт к тупику, т.к. процессы могут не запрашивать макс. необходимое им кол-во единиц ресурса.

Опр. Сост-е явл. безопасным, если сущ. такая посл-ть процессов, что
- 1-й процесс в посл-ти завершится, т.к. даже если он запросит макс. заявленное кол-во ед. ресурса, система имеет необходимое кол-во ед. ресурса для удовлетворения запроса;
- 2-й процесс в посл-ти может завершиться, если завершится 1-й и вернёт системе все затребованные им ед. ресурса, что в сумме со свободными ед. сможет удовлетворить макс. потребность 2-го процесса;

– i-й процесс в посл-ти может завершиться, если все i-1 процессов завершились успешно, освободили захваченные ресурсы и в сумме освобождённые и свободные ед. ресурса могут удовлетворить запрос i-го процесса, даже если он запросит макс. необх. кол-во ед. ресурса;

Т.о. всякий раз, когда процесс делает запрос, менеджер ресурсов должен найти успешно завершающуюся посл-ть процессов, и только в этом случае запрос м.б. удовлетворён;

Для определения состояния необходимо исследовать n! посл-тей, поэтому данный алгоритм имеет лишь теоретическое значение:
1) n! – это слишком большой объём вычислений;
2) для реализации алгоритма требуются очень сильные ограничения (фикс. кол-во процессов и ресурсов);

Но сама идея очень важна для систем, т.к. существуют системы (прежде всего ОСРВ), которые не могут попадать в тупики;
Да и вообще тупиковая ситуация для системы – не благо;

Решение Дейкстры – отправная точка для появления алгоритмов, описывающих затраты на анализ ситуации (опасная / безопасная);

Затраты м. б. снижены до n^2;

Алгоритм Хабермана (Habermann)

Общее опр-е алгоритма:

Менеджер ресурсов по алгоритму Хабермана поддерживает массив $S[0..r-1]$, где r – число единиц ресурса.

$S[i] = r - i$ для $\forall i: 0 \le i < r$; → (claim, заявка)

если процесс, заявивший C единиц ресурса и удерживающий h единиц ресурса, запрашивает ещё одну единицу ресурса, то $S[i]$ декрементируется для $\forall i; 0 \le i < C - h$;

Если какое-то из $S[i] < 0$, то такое сост-е системы будет опасным отн. тупика;

Графовая модель Хольта: двудольный напр. граф описывает ситуацию в системе

Обнаружение тупиков

Совр. требование к работе системы: система не ограничивается требованиями, процессы создаются по мере необходимости, ресурсы им выделяются по мере надобности;

Тупики возможны ⇒ ставится задача их обнаружения.

Используется графовая модель Хольта: двудольный направленный граф:

- 2 непересекающихся подм-в вершин: процессы и ресурсы;
- вершины соединяются дугами, которые имеют направления;
- если дуга выходит из вершины подм-ва вер-

или процессов и входит в вершину подмн-ва вершин ресурсов, то такая дуга наз. запрос;
- если наоборот, то такая дуга наз. выделение;

При этом процессы постоянно делают запросы на ресурсы и, если это возможно, ресурсы выделяются процессам, т.е. состояние графа постоянно меняется;

При этом, если процесс не может поменять своего состояния ни путём запроса, ни путём последующего приобретения в рез-те запроса ресурса, ни путём освобождения ресурса, то такой процесс находится в тупике;

Обнаружить процессы, попавшие в тупик, можно методом редукции графа, т.е. сокращением графа:
Граф сокращается процессом P_i, если процесс в графе не явл. ни заблокированным, ни изолированной вершиной, путём удаления всех инцидентных ей дуг;

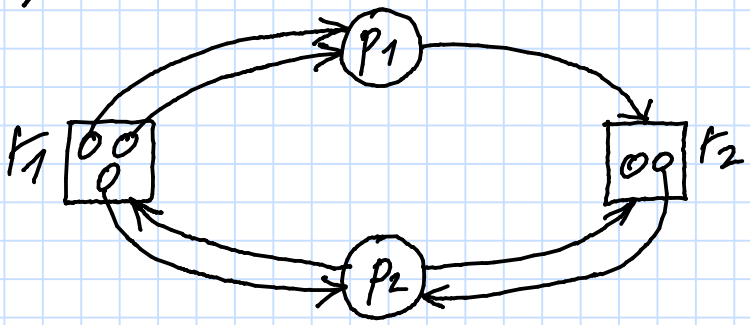
Процедура сокращения графа по вершине P_i соответствует действию процесса по запросу и возможному получению им запрошенного ресурса с последующим освобождением;

Процессы запрашивают ресурсы для того, чтобы иметь возможность продолжить своё выполнение;

если процесс не может продолжить своё выполнение, то он не может изменить своё состояние и он находится в тупике;

Мы это видели на простейшем примере с семафорами;

Пример полностью сокращаемого графа:



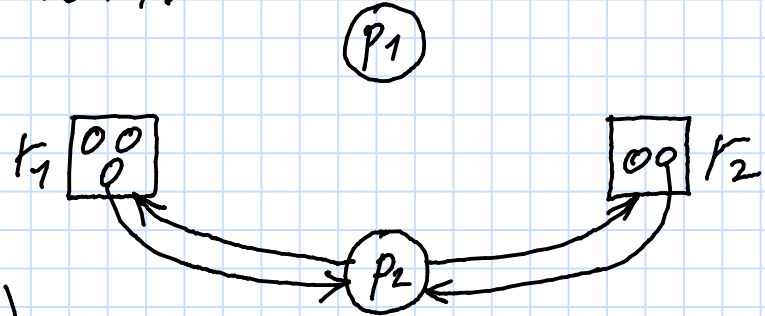
r_1, r_2 – типы ресурсов;

- P_1 ранее запросил и получил 2 ед. ресурса r_1;
- P_2 ранее запросил и получил 1 ед. ресурса r_1;
- P_2 ранее запросил и получил 1 ед. ресурса r_2;
- P_1 и P_2 запрашивают ещё по 1 ед. ресурса r_2;

Если процесс может получить запрошенную ед. ресурса, то он может продолжить своё выполнение, завершиться и освободить занимаемые им ед. ресурса;

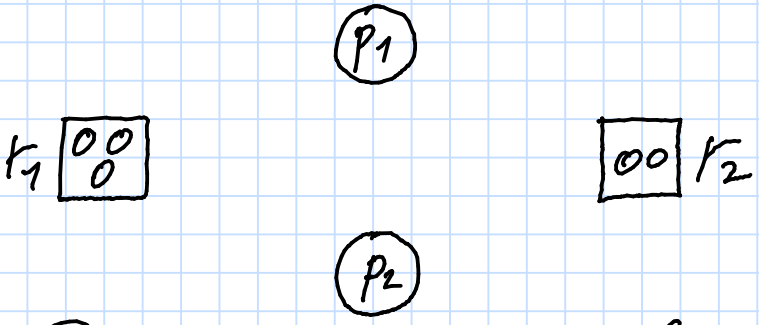
В данном случае существует посл-ть сокращений графа:

1) Сокращение по вершине P_1: у системы есть свободная ед. ресурса r_2, P_1 сможет её получить, завершиться и освободить 2 занимаемые ед. ресурса r_1:

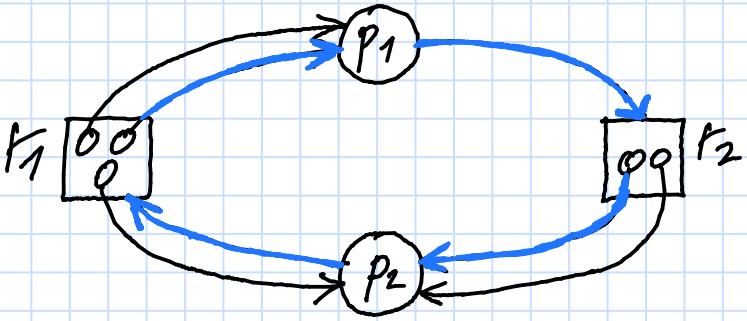


2) Теперь можем сократить по вершине P_2,

у системы имеется достаточное кол-во свободных ед. ресурса r_1 и ресурса r_2 $\Rightarrow$ граф явл. полностью сокращаемым:



Перестроим граф в несокращаемый:



Ни по p_1, ни по p_2 сократить нельзя;

Также здесь есть замк. цепь запросов;

Это есть в книге Шоу „Логическое проектирование ОС"

Теоремы из книги:

I) Граф явл. полностью сокращаемым, если сущ. такая посл-ть сокращений, которая устраняет все дуги;
Более строгое опр-е: состояние графа S есть состояние тупика $\Leftrightarrow$ граф повт. используемых ресурсов в S не явл. полностью сокращаемым;

II) Цикл (замкнутая цепь запросов) в графе повт. используемых ресурсов явл. необходимым условием тупика;

III) Если S не явл. состоянием тупика и переход $S \xrightarrow{i} T$ явл. тупиковым, то этот переход осуществляется в рез-те запроса;

Другими словами: тупик в системе может возникнуть только в рез-те запроса;

Т.е. если тупик возникает в рез-те запроса, то анализ состояния системы надо выполнять после каждого запроса;

Если для системы важно, чтобы она не оказалась в тупике, то выполняется не полный анализ состояния системы по графу (когда уже ничего нельзя сделать). А что тогда?

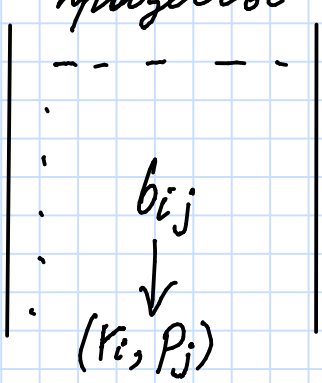
Что является внешними признаками того, что система находится в сост-ии тупика?

Система может работать с другим представлением графа. Например, в виде связных списков и/или матриц;

Связные списки сложно выписывать, поэтому остановимся на матрицах;

Матрица текущего распределения

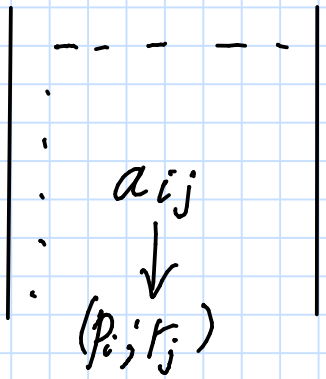
$B = \{r, p\}$



Эл-т матрицы b_{ij} отражает кол-во ед. ресурса r_i, выделенное процессу p_j;

Матрица запросов

$A = \{p, r\}$



эл-т матрицы a_{ij} отражает кол-во запрошенных ед. ресурса r_j процессом P_i;

Пример анализа ситуации, в которой находится система:

Матрица расп-я

P/r	1	2	3
1	0	1	1
2	1	3	0
3	1	5	0
	2	9	1

→ выделенные единицы ресурса

Матрица запросов

P/r	1	2	3
1	1	1	0
2	0	0	1
3	1	1	2

Вектор свободных единиц ресурса F

j	1	2	3
	4	0	2

Вектор совместно с уже выделенными ед. ресурса должен соотв общему кол-ву ед. ресурса в системе, т.е. должно выполняться следующее равенство:

$$R_j = \sum_{i=1}^{n} b_{ij} + F_j$$

В примере $\Sigma \rightarrow \underset{(4+2)}{6}\ \underset{(9+0)}{9}\ \underset{(1+2)}{3}$;

Сокращение графа может быть выполнено путём сравнения строки запросов с вектором своб. ресурсов; 2-й процесс единственный имеет строку запросов меньше, чем вектор своб. ресурсов по всем позициям:

	1	2	3
1	0	1	1
2	0	0	0
3	1	5	0

$\Rightarrow$ 2-й процесс может завершиться и мы можем сократить матрицу расп-я по 2-му процессу;
после завершения 2 процесс освободит занятые ресурсы $\Rightarrow$ вектор F будет равен (5; 3; 2);

Теперь может завершиться 1-й процесс, выполним сокращение по нему:

	1	2	3
1	0	0	0
2	0	0	0
3	1	5	0

Вектор своб. ресурсов: (6 9 3);

Теперь может завершиться 3-й процесс, выполним сокращение по нему:

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

$F = (6\ 9\ 3)$; - экв. кол-ву ресурсов в системе;

Граф оказался полностью сокращаемым;
Такое состояние системы не явл. тупиковым;

Это был т.н. прямой алгоритм обнаружения тупика, то есть более эффективный, который предполагает хранение доп. информации о запросах: для каждого ресурса хранятся запросы на него и упорядочиваются по размеру, а для каждого процесса определяется счётчик ожиданий, который содержит число типов ресурсов, вызвавших блокировку процесса в ожидании освобождения нужного ресурса;

Книга Медника и Донована:
алгоритм Bensusan-Murphy обнаружения тупиков для единичных ресурсов;

Графовая модель Холта представляется

в виде матриц. На этом построены практические методы анализа состояния системы.

Пример анализа сост-я системы по алгоритму Хобермана (явл. обобщением алгоритма Дейкстры):
(Как узнать, что сост-е системы явл. безопасным отн. тупика?)
Анализ сост-я системы по Хоберману:
1) анализ выполняем с заданного (текущего) состояния;
2) Процессы выполняются, при этом имитируется, что каждому процессу выделяется макс. кол-во запрошенных им ресурсов;
3) Если все процессы при этом завершатся, то состояние безопасное;

При этом рассматриваются n процессов и m ресурсов;
В системе имеются доступные ед. ресурса:

Матрица доступных ресурсов:

Max-Avail matrix $A = (a_1, a_2, \dots, a_m)$,
где $a_i = t_i = |R_i|$;

Матрица заявок:

Max-Claim matrix $B = \begin{vmatrix} b_{11} & \dots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nm} \end{vmatrix} = \begin{vmatrix} B_1 \\ \vdots \\ B_n \end{vmatrix}$;
(каждый процесс должен знать свою макс. потребность в ресурсах)

Матрица распределений:

Allocation matrix $C = \begin{vmatrix} c_{11} & \dots & c_{1m} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nm} \end{vmatrix} = \begin{vmatrix} C_1 \\ \vdots \\ C_n \end{vmatrix}$
(кол-во ед. ресурса r_j, выд. и удерживаемое процессом p_i.)

Ограничения:

1) Все k, $B_k \le A$ — процесс не может запрашивать ресурсов больше, чем имеется в системе;
2) $C \le B$ — процесс не может запрашивать ресурсов больше, чем есть в системе;
3) $\sum_{k=1}^{n} C_k \le A$ — никогда не выделяется больше ресурсов, чем доступно;
4) Вводится матрица доступных ресурсов

$$D = (d_1, \dots, d_m) = A - \sum_{k=1}^{n} C_k,$$

т.е. d_i — кол-во единиц ресурса r_i, доступное в текущем состоянии;

Появление более эффективного алгоритма выполнения данного анализа приводит к необходимости введения доп. матриц:

5) Матрица возможных запросов

$$\text{Need matrix } E = \begin{vmatrix} e_{11} & \dots & e_{1m} \\ \vdots & \ddots & \\ e_{n1} & & e_{nm} \end{vmatrix} = B - C = \begin{vmatrix} E_1 \\ \vdots \\ E_n \end{vmatrix};$$

6) Матрица запросов:

$$\text{Request matrix } F = \begin{vmatrix} f_{11} & \dots & f_{1m} \\ \vdots & \ddots & \\ f_{n1} & & f_{nm} \end{vmatrix} = \begin{vmatrix} F_1 \\ \vdots \\ F_n \end{vmatrix};$$

7) Предварительное состояние:

Предполагается, что
$D \leftarrow D - F_i$ // доступно - запросы;
$C_i \leftarrow C_i + F_i$ // распределено - запрошено;
$E_i \leftarrow E_i - F_i$ // требуется - запрошено;
в рез-те запрос будет удовлетворён, если предыдущее состояние было безопасным;

Алгоритм проверки безопасного состояния:

1) Выбираем незавершившийся процесс P_i такой, что $E_i \leq D$;
   Если такого процесса нет, переходим на шаг 3;
2) Если такой процесс есть, отмечаем P_i как завершившийся и переходим на шаг 1;
3) Если все процессы отмечены как завершившиеся, то система находится в безопасном состоянии, иначе в небезопасном;

При этом, если система находится в небезопасном сост-ии, то запрос процесса блокируется и состояние системы сбрасывается след. образом:

Сброс (по п. 7):
$D \leftarrow D + F_i$
$C_i \leftarrow C_i - F_i$
$E_i \leftarrow E_i + F_i$

При этом процесс, который был отмечен как завершённый на шаге 2, сбрасывается и отмечается как незавершившийся.

Числовой пример:

Max - Avail $A = (2\ 4\ 3)$;

Max - Claim $B = \begin{vmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$;

Allocation (распределено) $C = \begin{array}{c|ccc} & r_1 & r_2 & r_3 \\ p_1 & 1 & 2 & 0 \\ p_2 & 0 & 1 & 1 \\ p_3 & 1 & 0 & 1 \end{array}$;

Available (доступно) $D = (0\ 1\ 1) \Rightarrow (2\ 3\ 4) - \overset{A}{2\ 3\ 1} \Rightarrow 0\ 1\ 1$ $\left(\sum_1^3 C_k\right)$

Возможные запросы $E = \begin{vmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix} = B_i - C_i$;

Пусть p_1 делает запрос
$F_1 = (0\ 0\ 1)$ запрашивает единицу ресурса r_3

Предварительное состояние будет

Доступно $D = (0\ 1\ 0)$, т.к. p_1 взял единицу ресурса r_3;

Матрица распр-я C:

$C = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{vmatrix}$; т.к. p_1 запросил ед. ресурса r_3

Матрица возможных запросов:

$E = \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix}$;

В этом состоянии (после запроса процессом p_1 ед. ресурса r_3) процесс p_3 может завершиться, т.к. $E_3 \leq D$;

Завершившись, p_3 вернёт удерж. им ресурсы $(1\ 0\ 1)$ в пул свободных ресурсов $\Rightarrow D = (1\ 1\ 1)$;

В рез-те м.б. удовлетворить процессы p_1 и p_2 $\Longrightarrow$ предв. анализ состояния системы после запроса процесса p_1 указывает на то, что состояние системы будет безопасным;

Задача рассуждений: показать, что контроль состояния системы возможен, и совсем не обязательно сбрасывать состояние системы, перезагружать ее и т.д., т.к. зачастую это практически невозможно;

Единственное средство оценки состояния системы – таймауты;

В распределённых системах всё по-другому