Deadlock Avoidance

CSCI 3753 Operating Systems

Spring 2005

Prof. Rick Han

Announcements

- HW #4 is due Tuesday March 8, 11 am
  - extra office hours Monday March 7, 2 pm
- PA #2 assigned, due Thursday March 17 11:55 pm
- Midterm is Thursday March 10
  - covers chapter 1-10
  - Midterm review is Tuesday March 8
- Read chapter 10
- Section this week is grader and prof

From last time...

- We discussed:
  - Deadlock Modeling
  - 4 necessary conditions for deadlock
    - mutex
    - hold-and-wait
    - no preemption
    - circular waiting
  - Deadlock Prevention - ways to prevent at least 1 of these conditions from coming true

Deadlock Avoidance

- Each process provides OS with information about its requests and releases for resources R_i
  - OS decides whether deadlock will occur at run time
  - e.g. batch jobs know a priori which resources they'll request and when
  - weakness: need a priori info
  - simple strategy: specify a *maximum* claim
    - knowing all future requests and releases is difficult
    - estimating a maximum demand for resources for each process is easier to produce
- A *resource allocation state* is defined by
  - # of available resources
  - # of allocated resources to each process
  - maximum demands by each process

Deadlock Avoidance

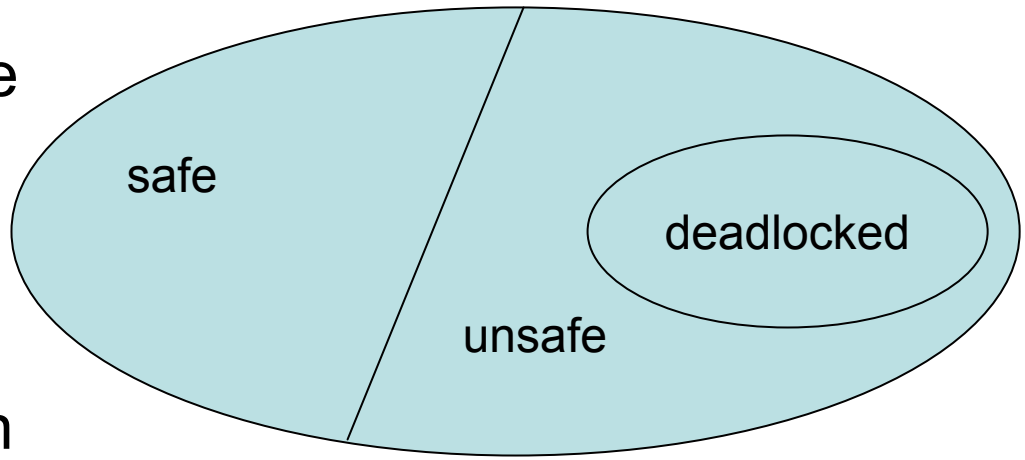
- A state is *safe* if there exists a *safe sequence* of processes <P1, ..., Pn> for the current resource allocation state
  - A sequence of processes is safe if for each P_i in the sequence, the resource requests that P_i can still make can be satisfied by:
    - currently available resources + all resources held by all previous processes in the sequence P_j, $j<i$
  - If resources needed by P_i are not available, P_i waits for all P_j to release their resources

Deadlock Avoidance

- Intuition for a safe state: given that the system is in a certain state, we want to find at least one "way out of trouble"
  - i.e. find a sequence of processes that, even when they demand their maximum resources, won't deadlock the system
  - this is a worst-case analysis
    - it may be that during the normal execution of processes, none ever demands its maximum in a way that causes deadlock
      - to perform a more optimal (less than worst-case) analysis is more complex, and also requires a record of future accesses

Deadlock Avoidance

- A safe state provides a safe “escape” sequence
- A deadlocked state is unsafe
- An unsafe state is not necessarily deadlocked
- A system may transition from a safe to an unsafe state if a request for resources is granted
  - ideally, check with each request for resources whether the system is still safe



Deadlock Avoidance

- Example 1:
  - 12 instances of a resource
  - At time t0, P0 holds 5, P1 holds 2, P2 holds 2
  - Available = 3 free instances

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	2

Deadlock Avoidance

- Example 1 (cont)
  - Is the system in a safe state? Can I find a safe sequence?
  - Yes, I claim the sequence <P1, P0, P2> is safe.
    - P1 requests its maximum (currently has 2, so needs 2 more) and holds 4, then there is only 1 free resource
    - Then P1 then releases all of its held resources, so that there are 5 free
    - Next, suppose P0 requests its maximum (currently has 5, so needs 5 more) and holds 10, so that there are 0 free
    - Then P0 releases all of its held resources, so that there are 10 free
    - Next P2 requests its max of 9, leaving 3 free and then releases them all
  - Thus the sequence <P1, P0, P2> is safe, and is able in the worst-case to request maximum resources for each process in the sequence, and release all such resources for the next process in the sequence

Deadlock Avoidance

- Example 2:
  - at time t1, process P2 requests and is given 1 more instance of the resource, then

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	3

  - Available = 2 free instances
  - Is the system in a safe state?

Deadlock Avoidance

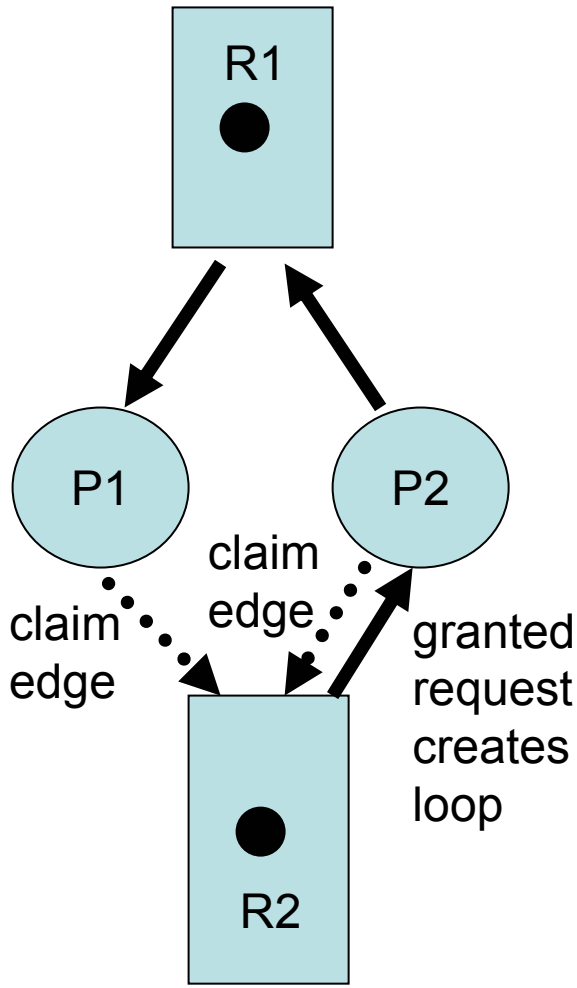
- Example 2 (cont)
  - P1 can request its maximum (currently holds 2, and needs 2 more) of 4, so that there are 0 free
  - P1 releases all its held resources, so that available = 4 free
  - Neither P0 nor P2 can request their maximum resources (P0 needs 5, P2 needs 6, and there are only 4 free)
    - Both would have to wait, so there could be deadlock
  - The system is deemed unsafe
    - The mistake was granting P2 an extra resource at time t1
    - Forcing P2 to wait for P0 or P1 to release their resources would have avoided potential deadlock

Deadlock Avoidance

- Policy: before granting a request, at each step, perform a worst-case analysis - is there a safe sequence, a way out?
  - If so, grant request.
  - If not, delay requestor, and wait for more resources to be freed.
- Banker's Algorithm takes this approach
- Resource allocation graph approach does not use a worst-case analysis (next slide), but is limited in its applicability

Deadlock Avoidance

- Using a Resource Allocation Graph for deadlock avoidance
  - each process identifies possible future claims, drawn as claim edges (dotted lines)
  - If converting a claim edge to a request edge creates a loop, then don't grant request
    - limited applicability - only valid when there is 1 instance of each resource
  - example: Granting P2's request for R2 creates a loop, so don't grant it



Deadlock Avoidance

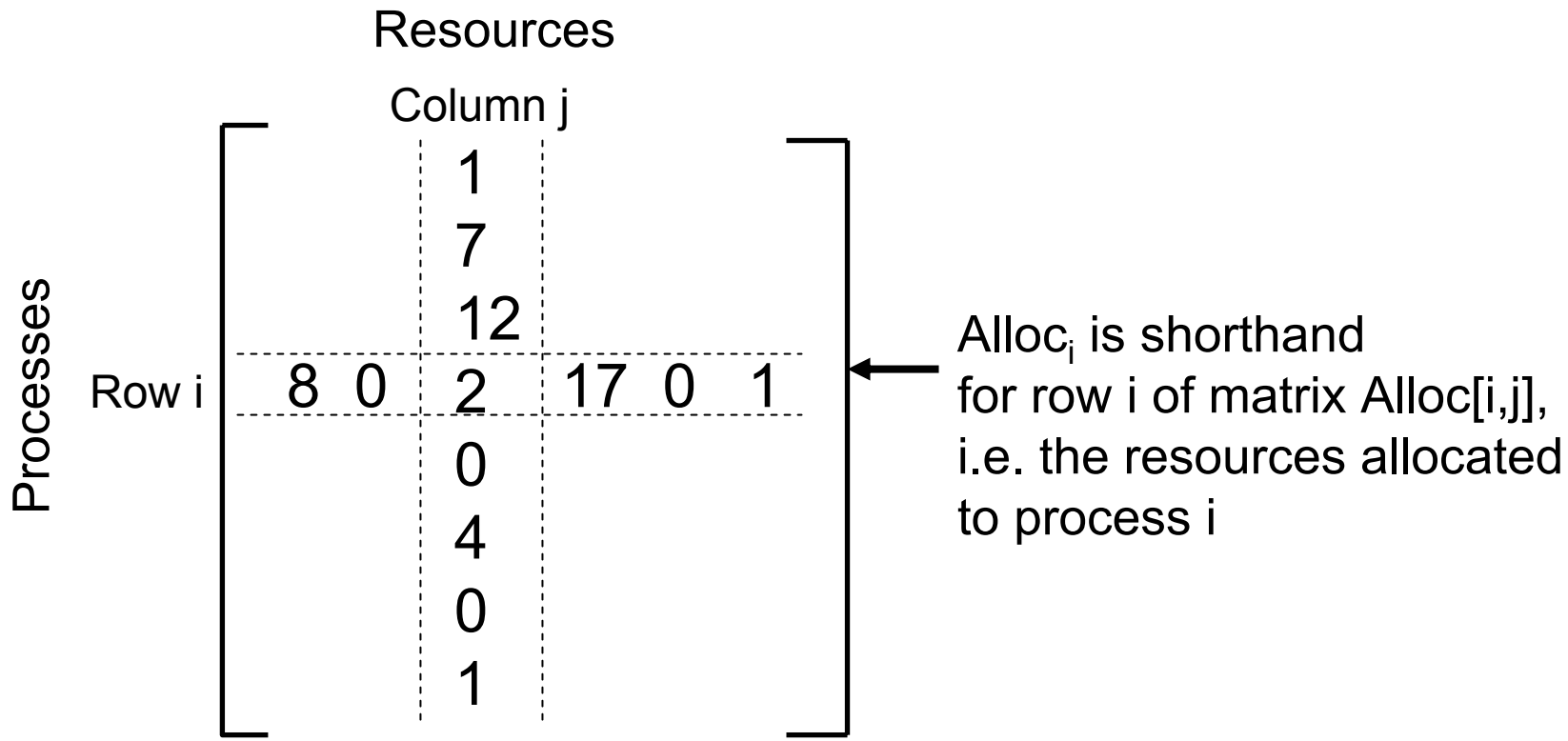
- Banker's Algorithm:
  - when there is a request, the system determines whether allocating resources for the request leaves the system in a safe state that avoids deadlock
    - if no, then wait for another process to release resources
  - each process declares its maximum demands
    - must be less than total resources in system
  - Advantages:
    - generalization of earlier examples (1 and 2), to allow multiple types of resources
    - works for multiple instances of resources, unlike resource allocation graph
    - adapts at run time to individual requests, avoids deadlock
    - individual requests don't need to be known a priori, except for maximum demands, which is not completely unrealistic

Deadlock Avoidance

- Banker's Algorithm:
  - Define quantities:
    - Available resources in a vector or list Available[j], j=1,...,m resource types
      - Available[j] = k means that there are k instances of resource R_j available
    - Maximum demands in a matrix or table Max[i,j], where i=1,...,n processes, and j=1,...,m resource types
      - Max[i,j] = k means that process i's maximum demands are for k instances of resource R_j
    - Allocated resources in a matrix Alloc[i,j]
      - Alloc[i,j] = k means that process i is currently allocated k instances of resource R_j
    - Needed resources in a matrix Need[i,j] = Max[i,j] - Alloc[i,j]

Deadlock Avoidance

- Banker's Algorithm:
  - An example of the Alloc[i,j] matrix:



Deadlock Avoidance

- Banker's Algorithm:
  - Some terminology:
    - let X and Y be two vectors. Then we say X ≤ Y if and only if X[i] ≤ Y[i] for all i.
    - Example:

$$V1 = \begin{bmatrix} 1 \\ 7 \\ 3 \\ 2 \end{bmatrix} \qquad V2 = \begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} \qquad V3 = \begin{bmatrix} 0 \\ 10 \\ 2 \\ 1 \end{bmatrix}$$

then V2 ≤ V1, but V3 ≰ V1, i.e. V3 is not less than or equal to V1

Deadlock Avoidance

- **Banker's (Safety) Algorithm**: find a safe sequence, i.e. is the system in a safe state?
  1. Let Work and Finish be vectors length m and n respectively. Initialize Work = Available, and Finish[i]=false for i=0,...,n-1
  2. Find a process i such that both
     - Finish[i]==false, and
     - $Need_i \leq Work$

     If no such i exists, go to step 4.
  3. Work = Work + $Alloc_i$ ←

     Finish[i] = true

     Go to step 2.
  4. If Finish[i]==true for all i, then the system is in a safe state

Intuition: if all prior processes give up all their resources, is there enough to meet the max needs of next process in the sequence?

Deadlock Avoidance

- Example 3:
  - 3 resources (A,B,C) with total instances available (10,5,7)
  - 5 processes
  - At time t0, the allocated resources Alloc[i,j], Max needs Max[i,j], and Available resources Avail[j], are:

Alloc[i,j]

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max[i,j]

	A	B	C
	7	5	3
	3	2	2
	9	0	2
	2	2	2
	4	3	3

Avail[j]

A	B	C
3	3	2

Need[i,j]

A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

where Need[i,j] is computed given Alloc[i,j] and Max[i,j]

Deadlock Avoidance

- Example 3 (cont)
  - Execute Banker's Algorithm - is the system in a safe state? is there a safe sequence?
  - Yes, I claim the sequence <P1, P3, P4, P2, P0> is safe
    - Both P1 and P3 have needs that can be satisfied by what is available. Select P1.
      - P1's $Need_1$ = <1,2,2> ≤ Available = <3,3,2>
      - P1 takes all that it needs, then releases all held resources, which equals its maximum demands. So update

        Available' = Available -$Need_1$ + Max_1

        = Available + $Alloc_1$, which we're tracking with Work in the algorithm

        = <3,3,2> + <2,0,0> = <5,3,2>
    - Next select P3 whose $Need_3$ = <0,1,1> ≤ Work = Available' = <5,3,2>
      - P3 takes all it needs, and releases its maximum claim, so
        - » Work = Available '' = Available' + $Alloc_3$ = <5,3,2> + <2,1,1> = <7,4,3>
    - If we look at the remaining process's needs P0, P2, and P4, we see that they all satisfy $Need_i$ ≤ Work = Available'', so we can satisfy their needs in any order, say P4, P2, P0. Thus, the sequence <P1, P3, P4, P2, P0> is safe.

Deadlock Avoidance

- Complexity of Banker's Algorithm is order $O(m*n^2)$
  - If there are n processes, then in the worst case the processes are ordered such that each iteration of the banker's algorithm must evaluate all remaining processes before the last one satisfies $Need_i \leq Work$
    - the 1^{st} time, n processes have to compare if their $Need_i \leq Work$
    - the 2^{nd} time, n-1 processes have to be compared
    - Thus, in the worst case, there are $n + (n-1) + (n-2) + ... + 2 + 1 = n(n+1)/2$ comparisons, which is proportional to n^2 complexity
    - Each vector comparison requires m individual comparisons, since there are m resource types, so total complexity is $O(m*n^2)$

Deadlock Avoidance

- Questions about Banker's Algorithm:
  - If it finds a safe sequence, are there other safe sequences, i.e. is the safe sequence that is found unique?
    - The safe sequence is not necessarily unique. There may be others.
      - Simple case is if two processes P1 and P2 both can have their max needs satisfied by available resources, then one safe sequence is <P1, P2>, and another safe sequence is <P2, P1>

Deadlock Avoidance

- Questions about Banker's Algorithm (cont):
  - If a safe sequence is not found, could we reorder our search to find another safe sequence? do we have to search exhaustively all possible sequences and/or backtrack our search?
    - I haven't found a clear answer in all the information sources I've looked at
    - Here's my best guess: no amount of reordering will find a safe sequence if a safe sequence couldn't be found the first time, so we don't have to search exhaustively or backtrack
      - This is because each step of the algorithm only releases more resources. At each step, I am expanding the number of processes whose needs can be met by what has been cumulatively released by the previous processes. The order of process releases that we reach a given quantity of released (available) resources does not matter.
      - example: suppose there are 3 processes. We run B.A. and it selects P0 first whose max needs can be met (acquires and releases its max resources, increasing the number of available resources). Now suppose neither P1 nor P2 can satisfy their max needs. So we have not found a safe sequence yet. If we run the banker's algorithm again, and try to select P1 or P2 instead of P0, we see that neither P1 nor P2 could possibly have their needs met since there are even fewer resources than before (P0 has not released its currently held resources). Thus my claim is that if B.A. can't find a safe sequence the first time, then no amount of reordering will find another safe sequence.