

## Deadlocks

- System contains finite set of resources
  - memory space
  - printer
  - tape
  - file
  - access to non-reentrant code
- Process requests resource before using it, must release resource after use
- Process is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set

CMSC 412

1

## Formal Deadlocks

- 4 *necessary* deadlock conditions:
  - Mutual exclusion - at least one resource must be held in a non-sharable mode, that is, only a single process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource is released
  - Hold and wait - There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently held by other processors

CMSC 412

2

## Formal Deadlocks

- No preemption: Resources cannot be preempted; a resource can be released only voluntarily by the process holding it, after that process has completed its task
- Circular wait: There must exist a set  $\{P_0, \dots, P_n\}$  of waiting processes such that  $P_0$  is waiting for a resource that is held by  $P_1$ ,  $P_1$  is waiting for a resource held by  $P_2$  etc.
- Note that these are not sufficient conditions

CMSC 412

3

## Deadlock Prevention

- Ensure that one (or more) of the necessary conditions for deadlock do not hold
- Hold and wait
  - guarantee that when a process requests a resource, it does not hold any other resources
  - Each process could be allocated all needed resources before beginning execution
  - Alternately, process might only be allowed to wait for a new resource when it is not currently holding any resource

CMSC 412

4

## Deadlock Prevention

- Mutual exclusion
  - Sharable resources do not require mutually exclusive access and cannot be involved in a deadlock.
- Circular wait
  - Impose a total ordering on all resource types and make sure that each process claims all resources in increasing order of resource type enumeration
- No Preemption
  - virtualize resources and permit them to be preempted. For example, CPU can be preempted.

CMSC 412

5

## Deadlock Avoidance

- Require additional information about how resources are to be requested - decide to approve or disapprove requests on the fly
- Assume that each process lets us know its maximum resource request
- Safe state:
  - system can allocate resources to each process (up to its maximum) in *some order* and still avoid a deadlock
  - A system is in a safe state if there exists a *safe sequence*

CMSC 412

6

## Safe Sequence

- Sequence of processes  $\langle P_1, \dots, P_n \rangle$  is a safe sequence if for each  $P_i$ , the resources that  $P_i$  can request can be satisfied by the currently available resources plus the resources held by all  $P_j, j < i$
- If the necessary resources are not immediately available,  $P_i$  can always wait until all  $P_j, j < i$  have completed

CMSC 412

7

## Banker's Algorithm

- Each process must declare the maximum number of instances of each resource type it may need
- Maximum can't exceed resources available to system
- Variables:
  - n is the number of processes
  - m is the number of resource types
    - Available - vector of length m indicating the number of available resources of each type
    - Max - n by m matrix defining the maximum demand of each process
    - Allocation - n by m matrix defining number of resources of each type currently allocated to each process
    - Need: n by m matrix indicating remaining resource needs of each process

CMSC 412

8

- Work is a vector of length m (resources)
  - Finish is a vector of length n (processes)
1. Work = Available; Finish = false
  2. Find an  $i$  such that Finish[i] = false and Need <sub>$i$</sub>  ≤ Work if no such  $i$ , go to 4
  3. Work += Allocation <sub>$i$</sub> ; Finish[i] = true; goto step 2
  4. If Finish[i] = true for all  $i$ , system is in a safe state

all elements  
in the vector  
are ≤

≤

Note this requires  $m \times n^2$  steps

CMSC 412

9

## Banker's Algorithm - Example

Three resources: A, B, C (10, 5, 7 instances each)

Consider the snapshot of the system at this time

	Alloc			Max			Avail			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

System is in a safe state, since the sequence <P1, P3, P4, P2, P0> satisfy the safety criteria.

CMSC 412

10

## Resource Request Algorithm

- (1) If  $Request_i \leq Need_i$  then goto 3
  - otherwise - the process has exceeded its maximum claim
- (2) If  $Request_i \leq Available$  then goto 3
  - otherwise process must wait since resources are not available
- (3) Check request by having the system pretend that it has allocated the resources by modifying the state as follows:
  - $Available = Available - Request_i$
  - $Allocation = Allocation + Request_i$
  - $Need_i = Need_i - Request_i$
- Find out if resulting resource allocation state is safe, otherwise the request must wait.

CMSC 412

11

## Deadlock Detection

- **Resource Allocation Graph**
  - Graph consists of vertices
    - type  $P = \{P_1, \dots, P_n\}$  represent processes
    - type  $R = \{R_1, \dots, R_m\}$  represent resources
  - Directed edge from process  $P_i$  to resource type  $R_j$  signifies that a process  $i$  has requested resource type  $j$
  - *request edge*
  - A directed edge from  $R_j$  to  $P_i$  indicates that resource  $R_j$  has been allocated to process  $P_i$
  - *assignment edge*

CMSC 412

12

## Deadlock Detection (cont.)

- Resource types may have more than one instance
- Each resource vertex represents a resource type.
- Each resource instance is of a unique resource type, each resource instance is represented by a “subvertex” associated with a resource vertex
  - (Silverschatz represents resource vertices by squares, resource instance “subvertices” by dots in the square. Process vertices are represented by circles)
- A request edge points to a resource vertex
- An assignment edge points from a resource “subvertex” to a process vertex

CMSC 412

13

## Resource Allocation Graph

- When a process  $P_i$  requests an instance of resource type  $R_j$ , a request edge is inserted into the resource allocation graph
- When the request can be fulfilled, the request edge is transformed into an assignment edge
- When the process is done using the resource, the assignment edge is deleted
- If the graph contains no cycles, no deadlock can exist

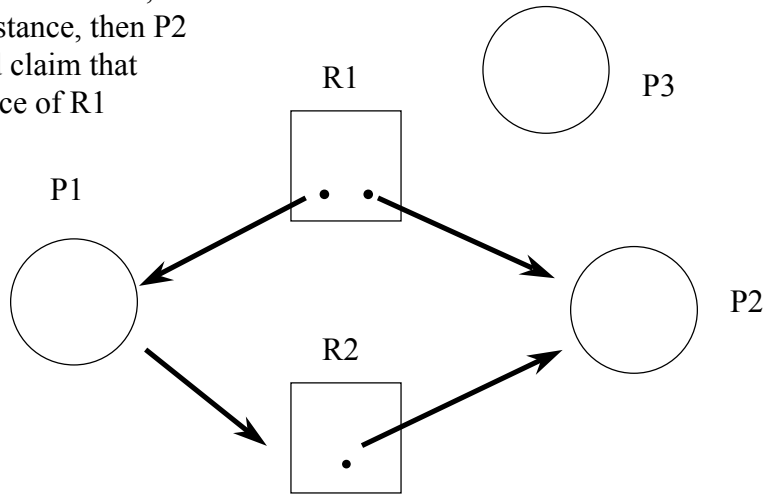
CMSC 412

14



P3 could finish with its instance of R1, release the instance, then P2 would claim that instance of R1

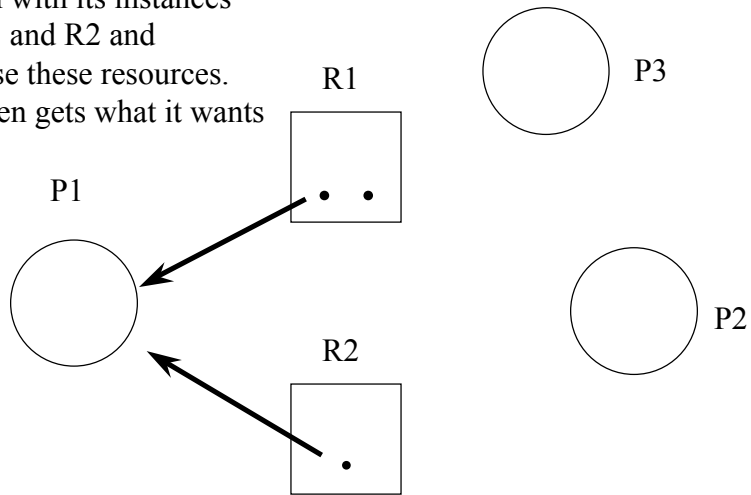
No!!



CMSC 412

17

Then, P2 could finish with its instances of R1 and R2 and release these resources. P1 then gets what it wants



CMSC 412

18

## Detecting Deadlock

Work is a vector of length  $m$  (resources)

Finish is a vector of length  $n$  (processes)

- Allocation is an  $n \times m$  matrix indicating the number of each resource type held by each process
- Request is an  $m \times n$  matrix indicating the number of additional resources requested by each process

1. Work = Available;

This is the difference from the Banker's algorithm.

if Allocation[i] != 0 Finish = false else Finish = true;

2. Find an  $i$  such that Finish[i] = false and Request <sub>$i$</sub>  <= Work if no such  $i$ , go to 4

3. Work += Allocation ; Finish[i] = true; goto step 2

4. If Finish[i] = false for some  $i$ , system is in deadlock

**Note: this requires  $m \times n^2$  steps**

CMSC 412

19

## Recovery from deadlock

- Must free up resources by some means
- Process termination
  - kill all deadlocked processes
  - select one process and kill it
    - must re-run deadlock detection algorithm again to see if it is freed.
- Resource Preemption
  - select a process, resource and de-allocate it
  - rollback the process
    - needs to be reset the process to a safe state
    - this requires additional state
  - starvation
    - what prevents a process from never finishing?

CMSC 412

20