4 points to cause a deadlock

Safe state:

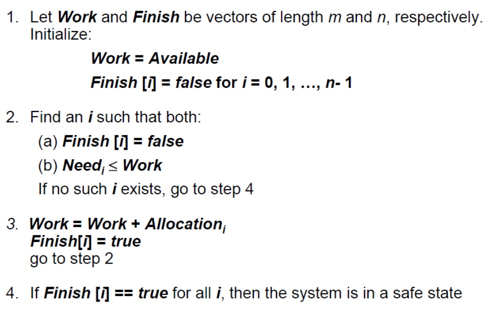
- No deadlocks will appear

Avoidance algorithm – for single instance of resource

Multiple resources and multiple instances

- Use banker's algorithm
  - Each process declares the max # of resources needed to finish
  - Can force a process to wait (decline giving a resource)
  - When process gets all resources must return them in a finite amount of time
    - Won't keep all resources forever

Assume # of processes and # of resources (of different types)



1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work = Available

   ***Finish [i] = false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:

   (a) ***Finish [i] = false***

   (b) ***Need**$_i$* $\leq$ ***Work***

   If no such ***i*** exists, go to step 4

3. ***Work = Work + Allocation**$_i$*
   Finish[i] = true
   go to step 2

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

Tensor (not matrix)

Max

- nxm matrix
  - Max # resources for each process

Allocation matrix

- Current allocation between processes and resources
- Define allocation to current process

Need

- Need = max – allocation
- How much left we need to complete the job
- We **compute** this matrix based on the other ones
- In banker algo
  - Has available, max for each processes, allocation for resources for different system processes

Close relationship between these 3 matrices

Safety Algorithm

- Two different vectors
  - Work is resources available at current time for all process running on system
  - Finish – boolean vector
    - False at start
      - No process finished its task
      - If a process finishes task, goes form false to true

Work is available resources on system

- Process completed task , if this conditional is true
  - Then it goes and sets finish to true

- If not completed
  - Then we allocate resources to a process

function of sequence of processes in certain order

- Never be in a non-safe state

If we go to stage 4 but finished == false

- We aren't finished, rather we are in a deadlock

Problem

- 5 processes at once
- 5 types of resources
  - Usb – a – 10 instances (10 possibilities to connect)
  - Terminal – 5 instances – b
  - Keyboard or RAM- c (7 instances)

Each process should claim max # resources for each type of resources

- 5 rows (processes), 3 columns (resources)
- We know alloc and max
  - Compute the need matrix
    - Max – allocation = need
    - Max # resources to complete task
      - P0 needs all these to complete task
        - Without that , wont complete
        - Hard constraint on system
          - How do we guess the future?
            - With dynamic allocation, etc we don't know

Work = available vector in beginning

Algo runs before we allocate

- Finds safe sequence of processes to allow it to complete
  - Safe sequence isn't unique
    - Multiple sequences can be valid

Find = search algorithm

- Limited
- 2nd step
  - Search from first or last?
    - Based on search can find different types of sequences

Algorithm

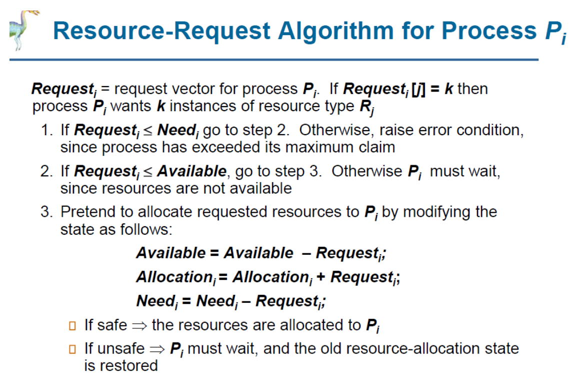
- Find could be unsuccessful for some search or get a different sequence or order

Prove system is not in a safe state?

- If we find a sequence (allows to run processes in certain order) for the processes = safe
- No sequence exists = unsafe

In reality

- Algorithm (run before we make the decision)
  - First finds order of processes on system
    - If found safe sequence
      - Then OS will follow the occurrence
        - As we are sure system won't be in a deadlock



Resource-Request Algorithm for Process P_i

Request**$_i$* = request vector for process ***P**$_i$*. If ***Request**$_i$* **[*j*] = *k then process ***P**$_i$* wants ***k*** instances of resource type ***R**$_j$*

1. If ***Request**$_i$* $\leq$ ***Need**$_i$* go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If ***Request**$_i$* $\leq$ ***Available***, go to step 3. Otherwise ***P**$_i$* must wait, since resources are not available
3. Pretend to allocate requested resources to ***P**$_i$* by modifying the state as follows:

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

- If safe $\Rightarrow$ the resources are allocated to ***P**$_i$*
- If unsafe $\Rightarrow$ ***P**$_i$* must wait, and the old resource-allocation state is restored

Sometimes one process at a time asks for request

- Now we need to use the request algorithm
  - Want to be sure that we can allocate for request and not be in a deadlock
    - 2 states

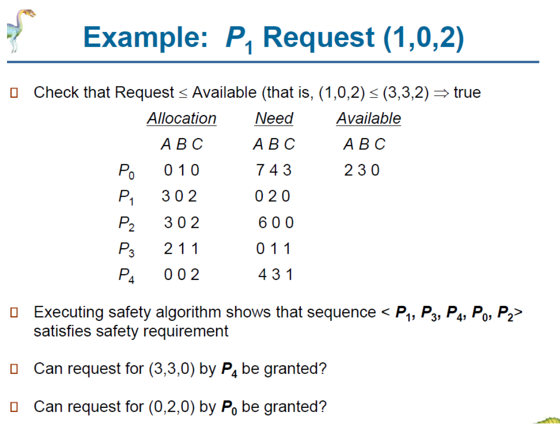
Request > than what we have = deadlock

If we accept request, remain in safe state

- Run the banker algorithm again to find safe state

Only satisfy request if we have enough resources

- If we deny the request, don't need to run the bankers algorithm
- **Necessary condition**
  - If we accept, are we in a safe state… still need to check the banker algorithm



Example: P_1 Request (1,0,2)

- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < $\boldsymbol{P_1, P_3, P_4, P_0, P_2}$> satisfies safety requirement
- Can request for (3,3,0) by $\boldsymbol{P_4}$ be granted?
- Can request for (0,2,0) by $\boldsymbol{P_0}$ be granted?

What changed:

- P1 requests 1 0 2 resources
  - Can accept
    - If we do , ensure that we still have a safe state

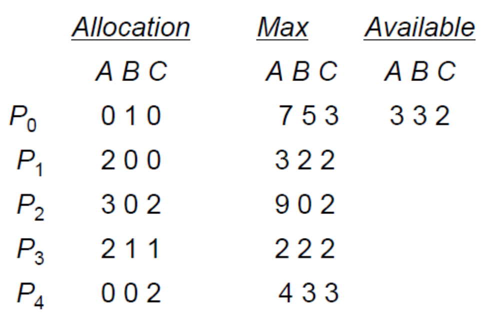
old allocation (OG BA)

can request (3,3,0) by p4, not enough resources

- So not accept

Can we accept (0,2,0) by p0

- Need to check req <= available
  - If so, need to check
    - Req <= need
      - If true then good



	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

New need = old max – new allocation (if we accept new request)

Deadlock detection = new algorithm

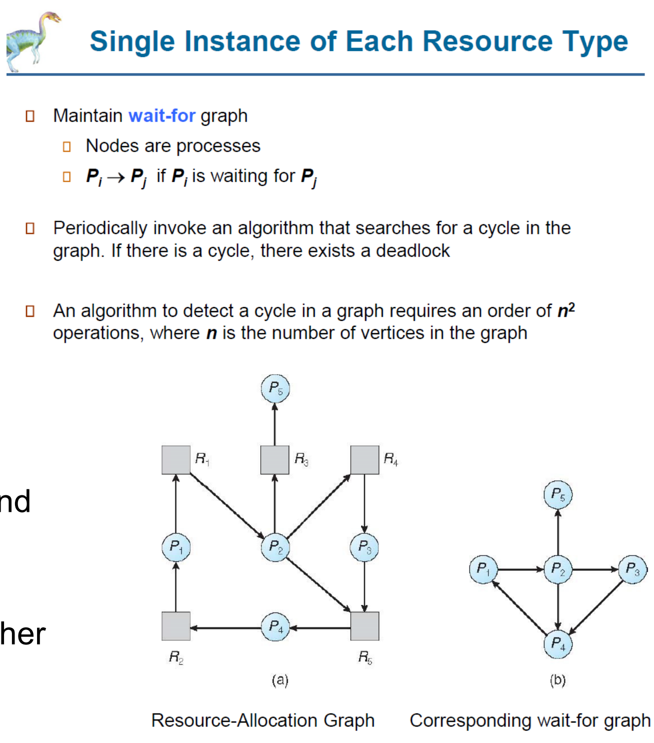
- Allow system to enter deadlock
  - Why?
- If yes, then detect if we are in deadlock or not
  - If we are = recovery scheme

circuluar wait = unsafe state

n^2 algorithm op

high # of processes on foreground and background

- Graph becomes complex
  - So use a wait for graph
    - When process waits for another one



Single Instance of Each Resource Type

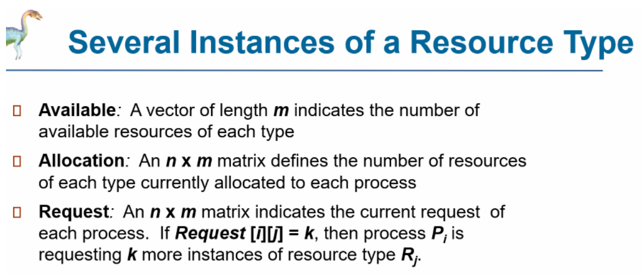
- Maintain **wait-for** graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

Resource-Allocation Graph Corresponding wait-for graph

Hard to find the graph on left and read it

Lets simplify the concept and detect deadlock via 2nd graph

- P4 waits for p1
  - P1 waits for p2
    - P2 waits for p3
      - Which waits for p4
        - Do we have a cycle – if yes (deadlock/unsafe)
          - Unsafe = includes both positions
            - As resource can have more than 1 instance
          - P1, p2, p4, p1
          - And a bigger cycle
            - P1,p2,p3,p4,p1
            - So system is in deadlock



Several Instances of a Resource Type

- **Available**: A vector of length m indicates the number of available resources of each type
- **Allocation**: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process
- **Request**: An $n \times m$ matrix indicates the current request of each process. If $Request[i][j] = k$, then process P_i is requesting k more instances of resource type R_j.

Detection algo

3 variables (for each process/type) resource

- Available
  - Each resource has x instances
- Allocation
- Requests

- Of all current processes

N = processes

M = resource #

- Each resource can have more than once instance

define 2 more vars

- Set finish value according to allocation
  - Not zero
    - Finish = false
      - Otherwise true
      - Different initialization state

Stage 2

- These conditions need to be agreed on, then go to next state

If doesn't have allocation, doesn't request, so assume it finished

- Each process completes task in certain amount of time

Get to safe state

- Order of sequences
  - 1b = shortcut / optimization

Stage 4

- Meaning is we are in a deadlock
  - Didn't satisfy condition of allocation
  - Unsafe state

goal is to get the same back In work as initial total

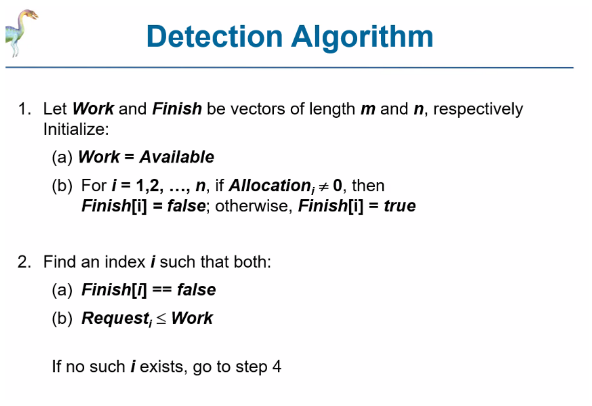
4th stage, CHECK AND VERIFY THAT FINISH IS TRUE FOR ALL processes

Similar to banker

- A process can ask for additional requests
- Cant allow p2 to run and get 1 byte more
  - Delays everything else
    - Computer crashes because of deadlock
      - Not safe state

Hard to do all checks

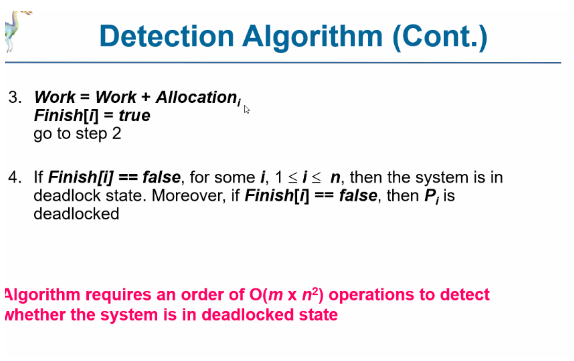
- Makes system heavy
- 2 types of OS
  - Unix



Detection Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:
   (a) ***Work = Available***
   (b) For ***i* = 1,2, …, *n***, if $\boldsymbol{Allocation_i \neq 0}$, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***
2. Find an index ***i*** such that both:
   (a) ***Finish*[*i*] == *false***
   (b) $\boldsymbol{Request_i \leq Work}$

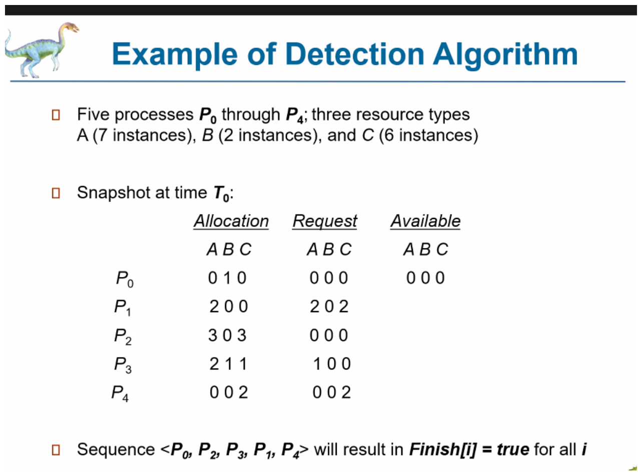
   If no such ***i*** exists, go to step 4



Detection Algorithm (Cont.)

3. $\boldsymbol{Work = Work + Allocation_i}$
   Finish*[*i*] = *true
   go to step 2
4. If ***Finish*[*i*] == *false***, for some ***i***, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if ***Finish*[*i*] == *false***, then P_i is deadlocked

Algorithm requires an order of O(*m* x *n*^{2}) operations to detect whether the system is in deadlocked state

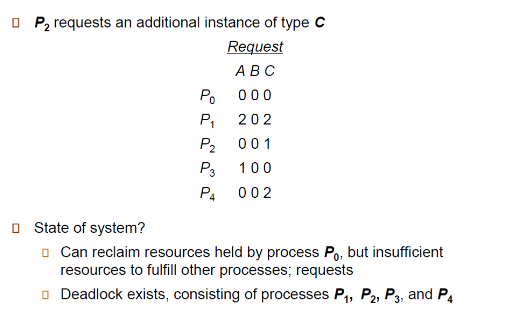


Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances)
- Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

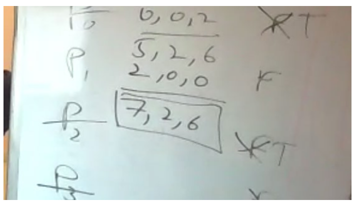
- Sequence <P_0, P_2, P_3, P_1, P_4> will result in ***Finish*[*i*] = *true*** for all ***i***



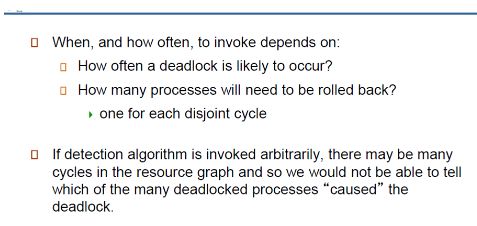
- P_2 requests an additional instance of type ***C***

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4



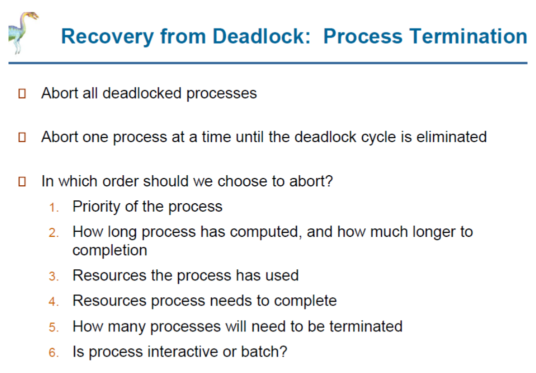
        - Probability is small, don't care
    - Windows
        - Can happen
            - Safe mode
            - Sure that every process on system can accomplish task
                - Slow and not easy
                    - Can solve computer issues
                        - Then return to the regular mode



- When, and how often, to invoke depends on:
    - How often a deadlock is likely to occur?
    - How many processes will need to be rolled back?
        - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from deadlock

- Abort processes
    - Fixes it
- Chose victim
    - Victim usually asks for lots of resources
        - Look at memory allocation
            - If small , don't kill him (doesn't pay off)
    - 103 processes running
        - Pick one that asks for lots of resources
    - But can be important and runs for a long time
        - So hard to say which victim to pick (depends on priority)
            - When all done, select victim process to perform task
                - Runs fast w/o problems



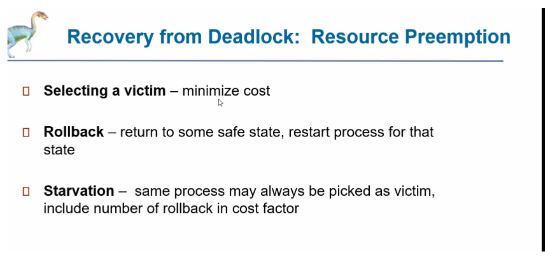
Recovery from Deadlock: Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
    1. Priority of the process
    2. How long process has computed, and how much longer to completion
    3. Resources the process has used
    4. Resources process needs to complete
    5. How many processes will need to be terminated
    6. Is process interactive or batch?

we define the cost based on task importance

- Based on load or importance
    - Select victim 0 return to safe state (no deadlock)
    - Rollback = restart process we select for that state
        - Don't always select same process as a victim = will be starved and never runs

  Select a victim
- Take p2 for example
    - Accept request and acceptance causes deadlock
        - So take p2, p1, or p3
            - P2 seems to be high priority process so maybe not a good idea
        - Maybe p3 or p4
            - We act on the Request part (add 1 more resource)
                - Type C resource
                    - They hold one or 2 instances of that type
                        - Overcome deadlock problem

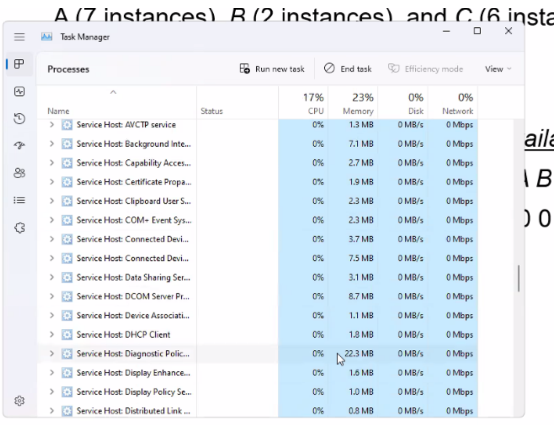


Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

Ensure to rollback the process we select

- Don’t take the same person always
-
- Look at cpu, disk, memory as resources
    - Don’t pick task manager to end as it causes other stuff to go awry (high cost)
        - We don’t need edge, lets kill it
            - But rerun it



Name	Status	17% CPU	23% Memory	0% Disk	0% Network
Service Host: AVCTP service		0%	1.3 MB	0 MB/s	0 Mbps
Service Host: Background Inte...		0%	7.1 MB	0 MB/s	0 Mbps
Service Host: Capability Acces...		0%	2.7 MB	0 MB/s	0 Mbps
Service Host: Certificate Propa...		0%	1.9 MB	0 MB/s	0 Mbps
Service Host: Clipboard User S...		0%	2.3 MB	0 MB/s	0 Mbps
Service Host: COM+ Event Sys...		0%	2.3 MB	0 MB/s	0 Mbps
Service Host: Connected Devi...		0%	3.7 MB	0 MB/s	0 Mbps
Service Host: Connected Devi...		0%	7.5 MB	0 MB/s	0 Mbps
Service Host: Data Sharing Ser...		0%	3.1 MB	0 MB/s	0 Mbps
Service Host: DCOM Server Pr...		0%	8.7 MB	0 MB/s	0 Mbps
Service Host: Device Associati...		0%	1.1 MB	0 MB/s	0 Mbps
Service Host: DHCP Client		0%	1.8 MB	0 MB/s	0 Mbps
Service Host: Diagnostic Polic...		0%	22.3 MB	0 MB/s	0 Mbps
Service Host: Display Enhance...		0%	1.6 MB	0 MB/s	0 Mbps
Service Host: Display Policy Se...		0%	1.0 MB	0 MB/s	0 Mbps
Service Host: Distributed Link ...		0%	0.8 MB	0 MB/s	0 Mbps

Same in linux too