ECE437/CS481

M05B: DEADLOCKS

CHAPTER 7

Xiang Sun

The University of New Mexico

Deadlock Handling

- Strategies to solve the deadlock problem
  - Prevention: design strategy that negates one of four deadlock necessary conditions.
  - Avoidance: dynamic strategy of processing resource requests to ensure that the system is always in a safe state
    - ✓ safe state --- if there exists at least one sequence of execution for all processes such that all of them can run to completion
    - ✓ state = safe state + unsafe state
    - ✓ unsafe state > deadlock state (sometimes it's unsafe, but not deadlock)
  - Detection and recovery: strategy for detecting the presence of deadlock and eliminating deadlock

- ❑ Deadlock Prevention
- ❑ Deadlock Avoidance
- ❑ Deadlock Detection and recovery

Deadlock Prevention

- Prevent the mutual exclusion condition from coming true
  - Recall mutual exclusion: at least 1 resource is held in a non-sharable mode. Other requesting processes must wait until the resource is released
  - This is opposite of our original goal, which was to provide mutual exclusion.
  - Also, many resources are non-sharable and must be accessed in a mutually exclusive way
    - ✓ example: a printer should print a file X to completion before printing a file Y. a printer should not print half of file X, and then print the first half of file Y on the same paper
  - thus, it is unrealistic to prevent mutual exclusion

Deadlock Prevention

- Prevent the hold and wait condition from coming true
  - Prevent a process from holding resources and requesting others
  - Solution I: request all resources at process creation
  - Solution II: release all held resources before requesting a set of new ones simultaneously
  - Example: a process reads from the DVD drive and writes the file to disk, sorts the file, then sends the file to the printer
    - ✓ Solution I: request the DVD drive, disk, and printer at process creation
    - ✓ Solution II: break the task down into wholly contained nondependent pieces:
      1. obtain the DVD and disk together for the file transfer, then release both together
      2. next obtain the disk and printer together for the printing operation, then release both together

Deadlock Prevention

- Disadvantages of Hold-and-wait solutions
  - Don't know in advance all resources needed
  - Poor resource utilization
    - ✓ a process that is holding multiple resources for a long time may only need each resource for a short time during execution
  - Possible starvation
    - ✓ a process that needs several popular resources simultaneously may have to wait a very long time

Deadlock Prevention

- Prevent the "No Preemption" condition from coming true
  - Allow resources to be preempted
  - Various policies are made for preemption, e.g.,
    - ✓ If a process X requests a resource held by process Y, then preempt the resource from process Y, but only if Y is waiting on another resource. Otherwise, X must wait.
    - ✓ If a Process X requests a held resource, then all resources currently held by X are released. X is restarted only when it can regain all needed resources.
  - Disadvantages
    - ✓ Preemption cannot be applied to all resources, e.g. printers should not be preempted while in the middle of printing, disks should not be preempted while in the middle of writing a block of data
    - ✓ Can result in unexpected behavior of processes, since an application developer may not know a priori which policy is being used.

Deadlock Prevention

- Prevent the circular wait condition from coming true

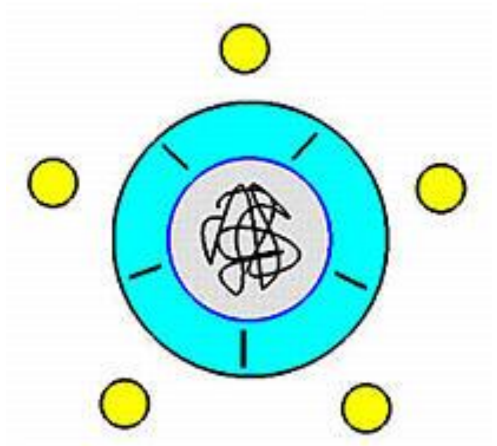
  - Basic idea: impose a total ordering of all resource types and require each process to request resources in increasing order

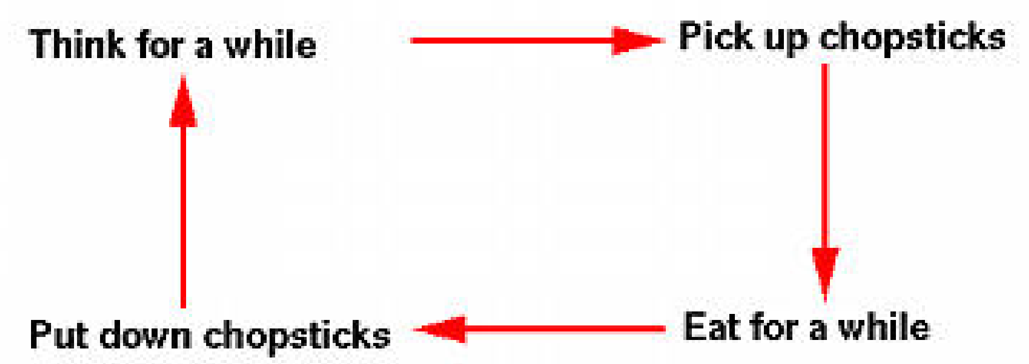
  - Example:

    - ✓ Order all resource types into a list: R_1, R_2, ..., R_m, where $R_1 < R_2 < \ldots < R_m$
    - ✓ Impose the rule that a process holding R_i can only request R_j if $R_j > R_i$
    - ✓ If a process holds R_k and requests R_j (where $R_j < R_k$), then the process must release all resources before R_k (including R_k), acquire R_j, and then reacquire R_k

Deadlock Prevention

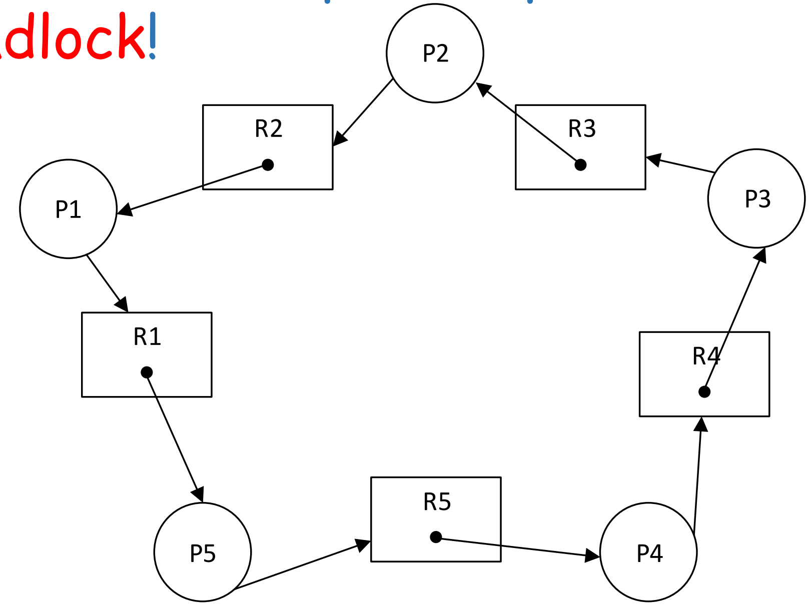
- Ordering of resource types to solve deadlock in the Dining Philosophers problem
  - Five philosophers sitting around a table. They spend their lives just thinking and eating.
  - There are five chopsticks available. When a philosopher get hungry, he tries to pick up two chopsticks that are closest to him.
  - A philosopher can eat only if he picks up two chopsticks.
  - Each chopsticks can only used by at most one philosopher at a time (mutual exclusion).
  - If a philosopher holds one chopstick but the other chopstick is currently used by another one, the philosopher will hold the chopstick and wait the other chopstick (hold & wait).
  - If a chopstick has been picked up by a philosopher, other philosophers cannot rob this chopstick (non-preemption).

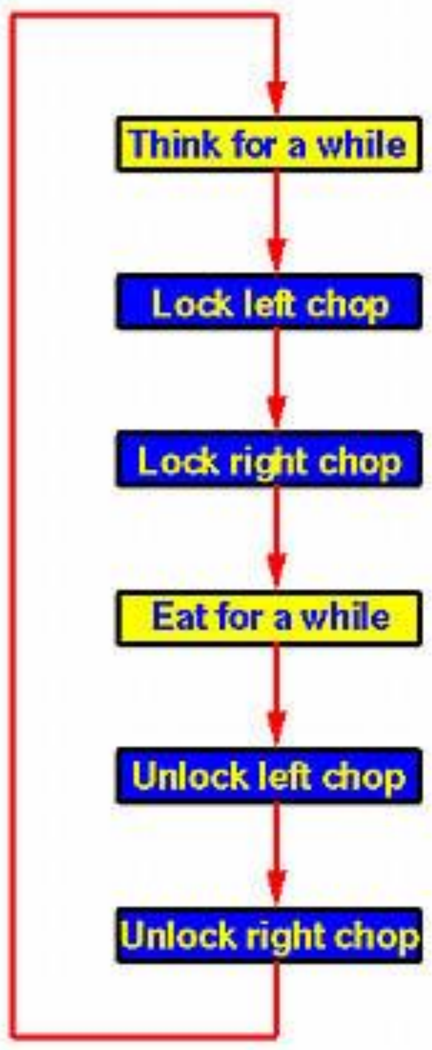




Deadlock Prevention

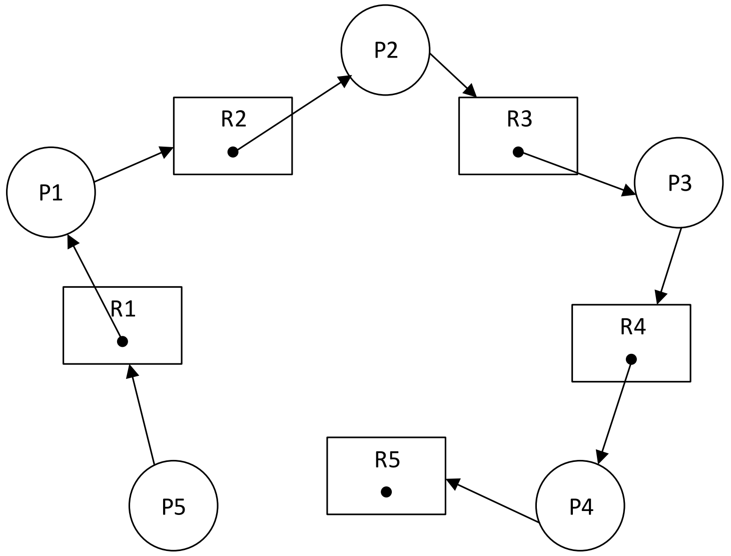
- Ordering of resource types to solve deadlock in the Dining Philosophers problem
  - In order to solve the problem of a chopstick picking up by two philosophers, each philosopher, before he can eat, locks his left chopstick and locks his right chopstick. If the acquisitions of both locks are successful, this philosopher now owns two locks (hence two chopsticks), and can eat. After finishes easting, this philosopher releases both chopsticks, and thinks.
  - What if five philosophers sits down and picks up their left chopsticks simultaneously?---deadlock!



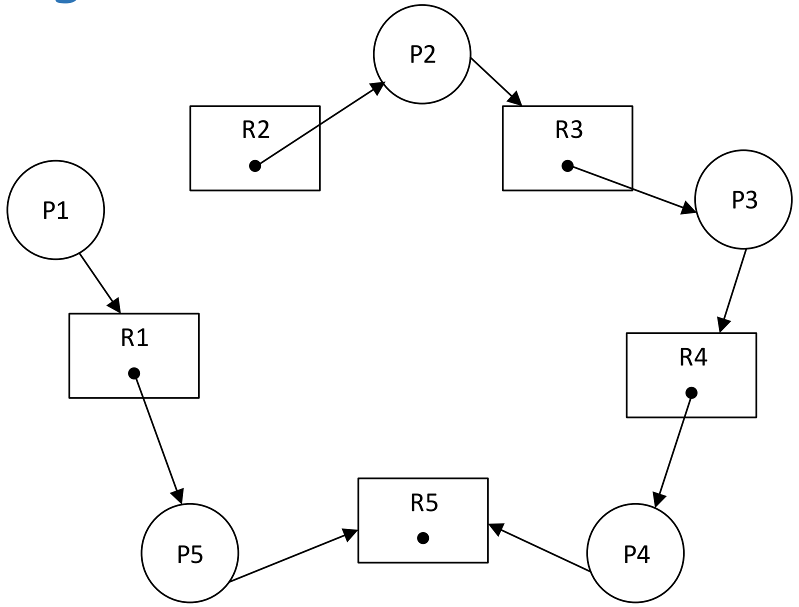


Deadlock Prevention

- Ordering of resource types to solve deadlock in the Dining Philosophers problem
  - R1 < R2 < R3 < R4 < R5
  - P1 requests R1, then R2, proceeding Right to Left
  - P2 requests R2, then R3, proceeding Right to Left
  - P3 requests R3, then R4, proceeding Right to Left
  - P4 requests R4, then R5, proceeding Right to Left
  - P5 requests R1, then R5, proceeding Left to Right



P1 hold R1 in the 1st round



P5 hold R1 in the 1st round

- ❑ Deadlock Prevention
- ❑ Deadlock Avoidance
- ❑ Deadlock Detection and recovery

Deadlock Avoidance

- ❑ Each process provides OS with the maximum amount of resource of each type that it may need.

- ❑ The deadlock-avoidance algorithm dynamically examines the resource allocation state to ensure that the system is in safe state (there can never be a circular-wait condition).

  - ➢ A resource allocation state is defined by
    - ✓ # of available resource instances
    - ✓ # of allocated resource instances to each process
    - ✓ maximum resource instances demand for each process

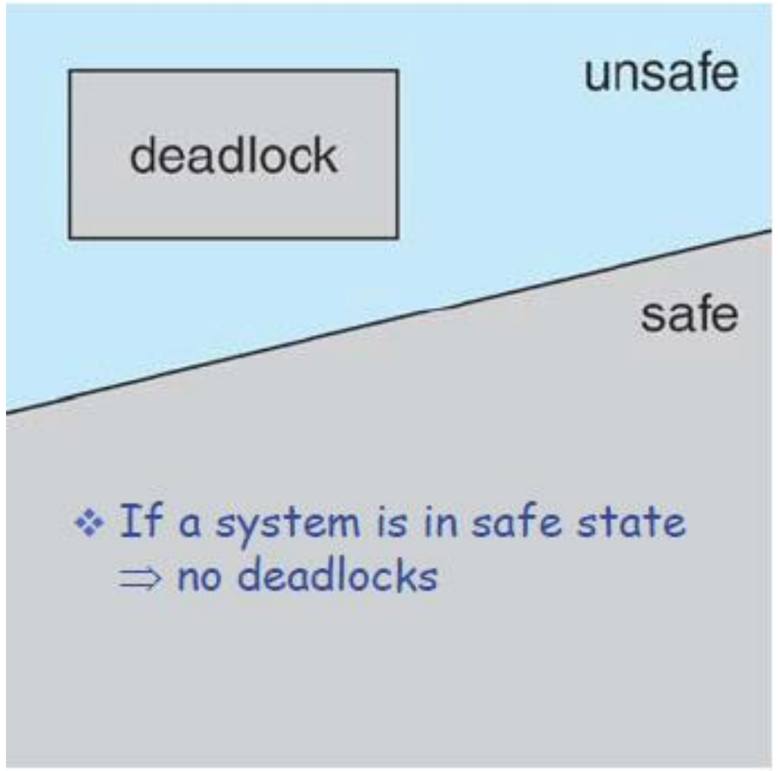
- ❑ Weakness of deadlock avoidance: need a priori info-- maximum amount of resource of each type

Deadlock Avoidance

- Definition of safe state: system is safe if there exists a safe sequence of processes $\langle P_1, \ldots, P_n \rangle$ for the current resource allocation state
  - A sequence of processes is safe---for each P_i in the sequence, the resource requests for P_i can be granted using the resources currently allocated to P_i and all processes P_j where $j < i$. (i.e. if all the processes prior to P_i finish and free up their resources, then P_i will be able to finish also, using the resources that they have freed up).
- That is
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

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



- If a system is in safe state $\Rightarrow$ no deadlocks

Deadlock Avoidance

- Example 1:
  - 12 instances of a resource
  - At time t_0, 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:

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

  - Is the system in a safe state? Does a safe sequence exist?
  - Yes. <P1, P0, P2> is a safe sequence.
    - ✓ 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 resource instances
    - ✓ Next, suppose P0 requests its maximum (currently has 5, so needs 5 more) and holds 10, so that there are 0 free resource instance
    - ✓ Then P0 releases all of its held resources, so that there are 10 free resource instances
    - ✓ 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 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 t_0, P2 is given 1 more instance of the resource (P2 holds 3 instances of the resource), then
  - Available = 2 free instances

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

  - Is the system in a safe state?

Deadlock Avoidance

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

- Example 2:
  - P1 can request its maximum (currently holds 2, and needs 2 more) of 4, so that there are 0 free resource instance.
  - P1 releases all its held resources, so that available = 4 free resource instance.
  - Neither P0 nor P2 can request their maximum resources (P0 needs 5, P2 needs 6, and there are only 4 free resource instance).

    ---Both would have to wait, so there could be deadlock

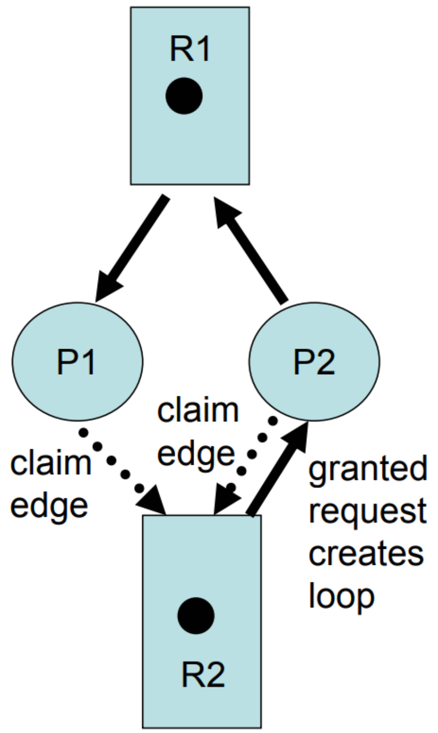
  - The system is unsafe.
    - ✓ The mistake was granting P2 an extra resource at time t1

Deadlock Avoidance

- Policy: before granting a request, at each step, perform the 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.
- Two methods of deadlock avoidance
  - Banker's algorithm follows the mentioned policy.
  - Resource allocation graph approach does not follow the policy, 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 the request.
  - Example: Granting P2's request for R2 creates a circle, so don't grant it.
  - Limited applicability - only valid when there is 1 instance of each resource



Deadlock Avoidance

- Banker's Algorithm:
  - Denote
    - ✓ i=1…n as index of processes.
    - ✓ j=1,…,m as the index of resource types.
    - ✓ Available[j] as the current available resource for resource type j. For example, Available[j]=5 indicates there are 5 instances of resource type j (i.e., Rj) available.
    - ✓ Max[i,j] as the maximum demands of process i for resource type j.
    - ✓ Alloc[i,j] as the number of resource instances that are currently allocated to process i for resource type j. For example, Alloc[i,j]=5 means 5 instances of resource type j are currently allocated to process i.
    - ✓ Need[i,j] as the number of resource instances of resource type j that are currently needed by process. Here, Need[i,j] = Max[i,j] -Alloc[i,j].

Deadlock Avoidance

❑ Banker's Algorithm: find a safe sequence, i.e. is the system in a safe state?

1. Let ***Work*** and ***Finish*** be vectors length m (number of resource types) and n (number of processes), respectively. Initialize ***Work*** = ***Available***, and Finish[i]=false for i=0,…,n-1

2. Find a process i such that
   - ✓ Finish[i]==false, and
   - ✓ Need[ij] ≤ Work[j], for all j //The needed resource of process i is less than the available resource

   If no such process i exists, go to step 4.

3. Work[j]=Work[j]+Alloc[ij]
   Finish[i] = True
   Go to Step 2.

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

Deadlock Avoidance

- Example of Banker's Algorithm
  - 3 resources (A,B,C) with total instances available (10,5,7)
  - 5 processes
  - Snapshot at time t0:

Work

Process	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Step 2:
Both P1 and P3 have needs that can be satisfied by what is available.

Deadlock Avoidance

❑ Example of Banker's Algorithm

- ➢ Let's allocate resources to P1 such that it can finish first.
- ➢ After P1 is finished, all the resources of P1 will be released.

Finish ***Work***

Finish	Process	Allocation	Max	Available (Work)	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	~~3 3 2~~ 5 3 2	7 4 3
True	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
	P_3	2 1 1	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

Step 2:
Both P3 and P4 have needs that can be satisfied by what is available.

Deadlock Avoidance

- Example of Banker's Algorithm

  - Let's allocate resources to P3 such that it can finish next.
  - After P3 is finished, all the resources of P3 will be released.

Finish	Process	Allocation	Max	Available (*Work*)	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	~~3 3 2~~ 7 4 3	7 4 3
True	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
True	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Step 2:
All P0, P2 and P4 can be satisfied by what is available.

Thus, the sequence <P1, P3, P0, P2, P4> is safe.

Deadlock Avoidance

- Summary of deadlock avoidance
  - a conservative strategy: process may have to wait even the resource requested is not in use
  - Drawbacks:
    1. Difficult to predict a process's max resource requirement.
       - ✓ giving a high estimate may result in long waiting time
    2. High runtime overhead
    3. All deadlock avoidance algorithms assume that processes are independent, that is, free from any synchronization constraint

- ❑ Deadlock Prevention
- ❑ Deadlock Avoidance
- ❑ Deadlock Detection and recovery

Detection and recovery

- Deadlock detection
  - Wait For Graph
  - Resource Allocation Graph
  - Banker's Algorithm (check if ***Finish(i)=="true"*** for all process i)
- Deadlock recovery
  - Recovery through preemption
    - ✓ take a resource from some other process
    - ✓ depends on nature of the resource
  - Recovery through rollback
    - ✓ checkpoint a process state periodically
    - ✓ rollback a process to its checkpoint state if it is found deadlocked
  - Recovery from process termination -- abort one process at a time until the deadlock cycle is eliminated. The other processes get its resources.

Detection and recovery

- Deadlock recovery
  - **Process termination** -- abort one process at a time until the deadlock cycle is eliminated.
  - Which process is selected to abort?
    - ✓ Priority of the process
    - ✓ How long the process has computed, and how long the process will be completed
    - ✓ Resources the process has used
    - ✓ Resources process needs to complete
    - ✓ Is process interactive or batch?