SWENG3043

Operating Systems

Software Engineering Department

AASTU

Dec 2018

Chapter Five

Deadlock

Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

Content

Overview

- Computer systems are full of resources that can only be used by one process at a time.
- Processes are granted exclusive access to devices. We refer to these devices generally as **resources.**
- Resource can be a hardware device(e.g. Memory, I/O devices, tape drive) or a piece of information (e.g., a locked record in a database).
- Sequence of events required to use a resource
  1. request the resource
  2. use the resource
  3. release the resource

Overview(cont'd)

- Suppose a process holds resource A and requests resource B
  - at same time another process holds B and requests A
  - both are blocked and remain so. This phenomenon is called deadlock.
- Resources types:
  - Preemptable resources
    - can be taken away from a process with no ill effects. E.g RAM
  - Nonpreemptable resources
    - will cause the process to fail if taken away. E.g. CD-recorder.
- In general deadlock involves non-preemptable resources

Overview(cont'd)

- Formal definition :
  - *A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.*
  - Usually the event is release of a currently held resource
- When deadlock occurs **none** of the processes can …
  - Run
  - release resources
  - be awakened
- **Examples:**
  - Suppose that process A asks for CD-ROM drive and gets it. A moment later, process B asks for a plotter and gets it, too. Now process A asks for the plotter and blocks waiting for it. Finally process B asks for CD-ROM drive and also blocks. This situation is called a deadlock.
  - Suppose process A locks record R1 and process B locks record R2, and then each process tries to lock the other one's record, this time a deadlock will happen.

Conditions for Deadlock

- A deadlock situation can arise if the following four conditions hold simultaneously:
  - **Mutual Exclusion:** only one process at a time can use the resource
  - **Hold and Wait Condition:** Processes currently holding resources granted earlier can request new resources
  - **No Preemptive Condition:** Resources previously granted cannot be forcibly taken away from a process
  - **Circular Wait Condition:** There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.
- If one of the four is absent, no deadlock is possible

Deadlock Modeling

- Deadlock conditions can be modeled using directed graph
- Two kinds of nodes: Process (circles) and Resources (square)
- An arc from a resource to a process means the resource is currently held by that process
- An arc from process to a resource means the process is currently blocked waiting for that resource

Deadlock Modeling(cont'd)

- **Example**: Let we have three processes A, B, C and three resources R1, R2 and R3. The request of the three processes are as follows:

  A

  - A1: requests R1
  - A2: requests R2

  B

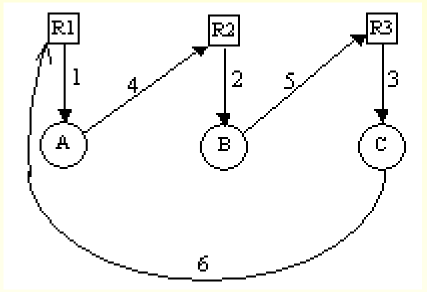
  - B1: Requests R2
  - B2: Requests R3

  C

  - C1: Requests R3
  - C2: Requests R1

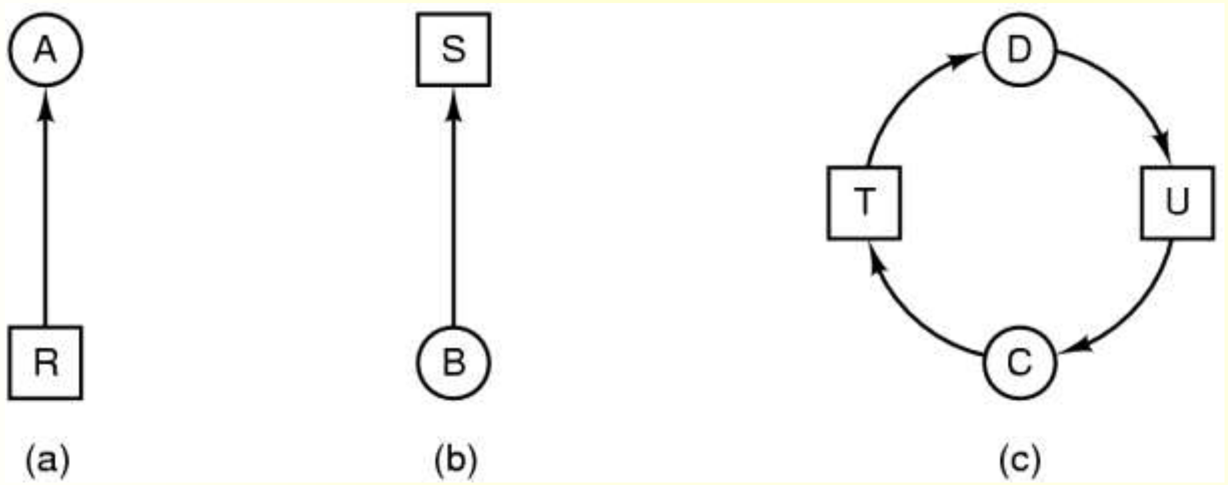
Deadlock Modeling(cont'd)

- Let round robin scheduling algorithm is used and instructions are executed in the following order:
  - A1→ B1→ C1 → A2 → B2 → C2



- Resource graphs are a tool that let us see if a given request/release sequence leads to deadlock
- If there is any cycle in the graph that means there is a deadlock

Deadlock Modeling(cont'd)



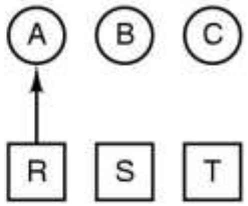
- resource R assigned to process A
- process B is requesting/waiting for resource S
- process C and D are in deadlock over resources T and U

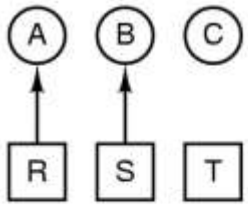
Deadlock Modeling(cont'd)

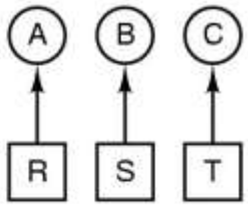
A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

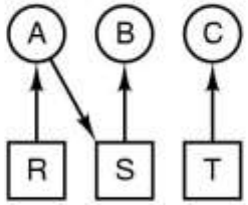
1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

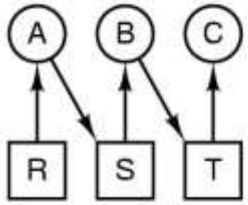
(d)

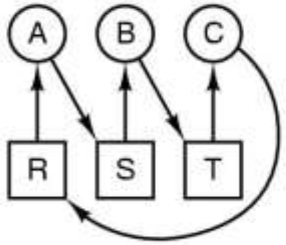

(e)


(f)


(g)


(h)


(i)

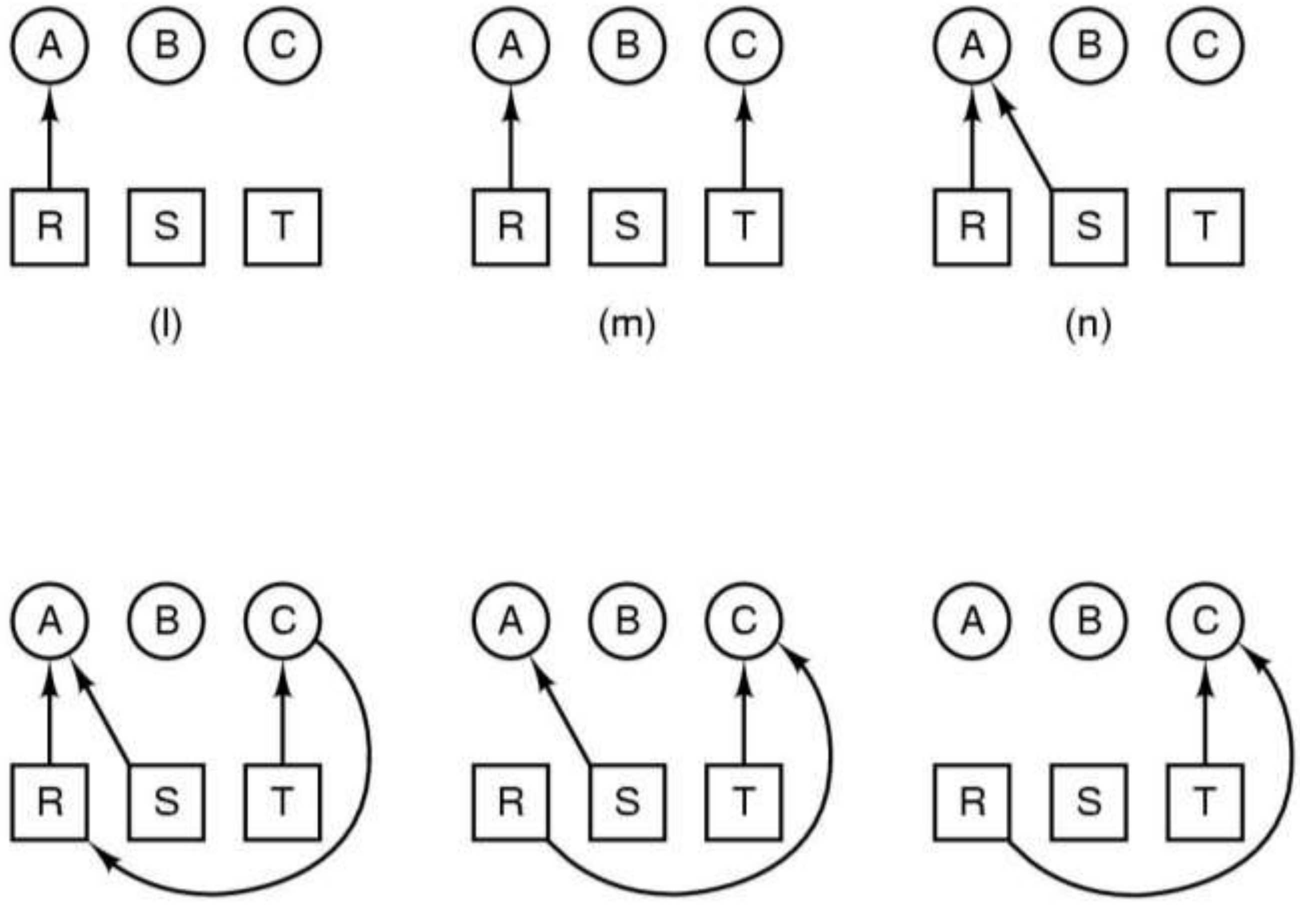

(j)

How deadlock occurs

Deadlock Modeling(cont’d)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock

(k)



(l) (m) (n)

(o) (p) (q)

How deadlock can be avoided

Methods for Handling Deadlocks

- In general, four strategies are used for dealing with deadlocks
  - Ignore the problem
  - Detect and recover
  - Dynamically avoid deadlock by careful resource allocation
  - Prevent, by structurally negating one of the four required conditions

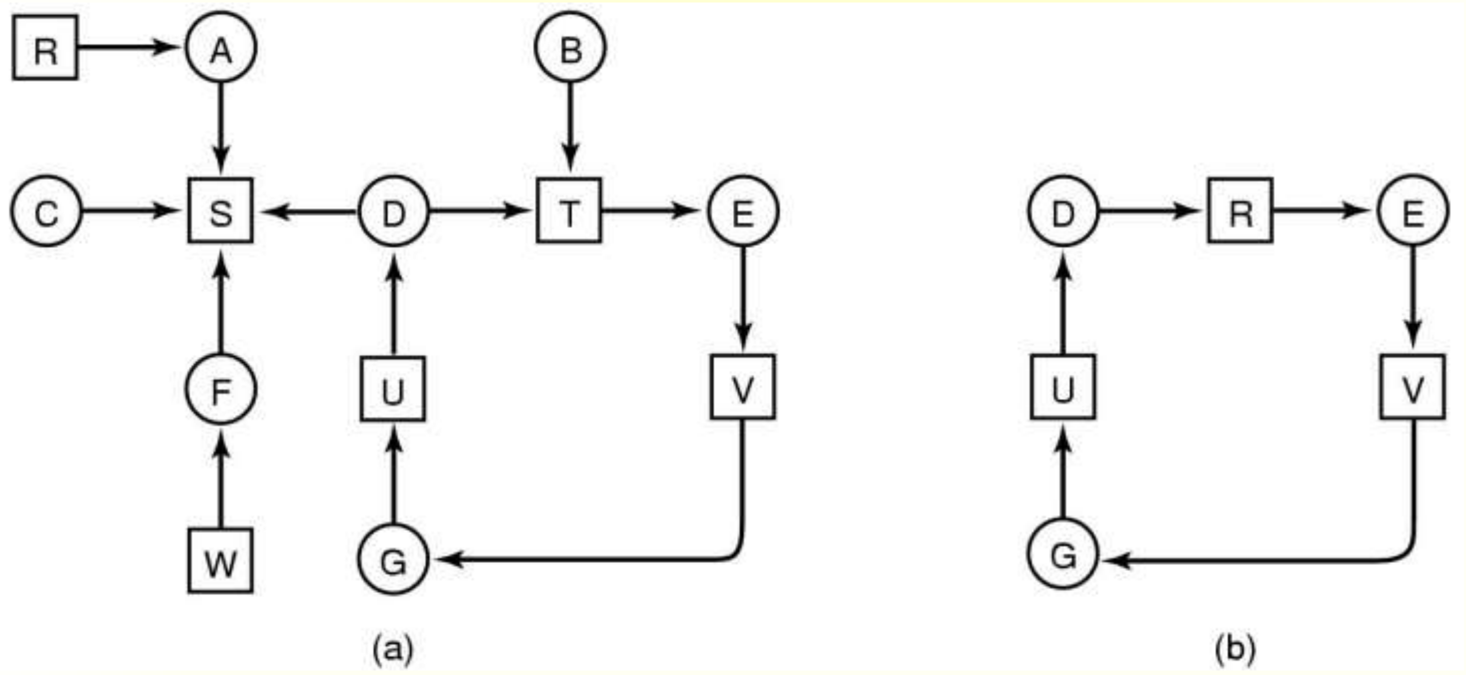
Ignore the problem

- **The Ostrich Algorithm**
  - It is the simplest approach
  - Stick your head in the sand and pretend there is no problem at all.
    - Mathematicians do not accept it
    - Engineers accept with constraints – frequency (chance of happening)
  - Reasonable if
    - deadlocks occur very rarely
    - cost of prevention is high
  - UNIX and Windows takes this approach

Detection and Recovery

- Every time a resource is requested or released, the resource graph is updated, and a check is made to see if any cycles exist.
- If cycle exists, one of the processes in the cycle is killed
- If the method is to check the length of time a process is requesting a resource and if large kill it

Detection and Recovery(cont'd)



- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock

Deadlock Recovery

- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked
- Recovery through killing processes
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning

Deadlock Avoidance

- We can avoid deadlocks, but only if certain information is available in advance
- Process declare the ***maximum number*** of resources of each type that it may need
- Construct an algorithm that ensures that the system will never enter a deadlocked state
- The Banker's algorithm considers each request as it occurs and sees if granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later.
- To see if a state is safe, the banker checks to see if he has enough resources to satisfy the customer closest to his or her maximum. If so, those loans are assumed to be repaid. If all loans can eventually be repaid, the state is safe and the initial request can be granted.

.

Safe state

A state is ***safe*** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

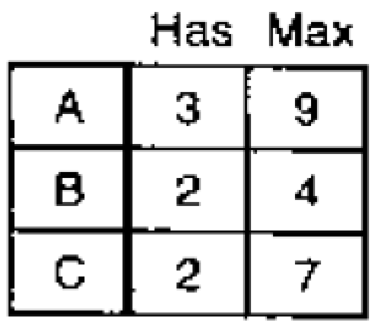
	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Demonstration that the state in (a) is safe

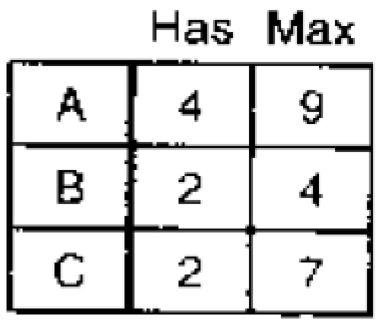
Unsafe state



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

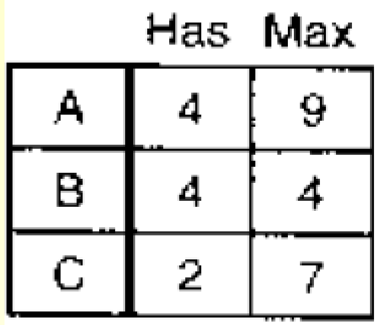
(a)



	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

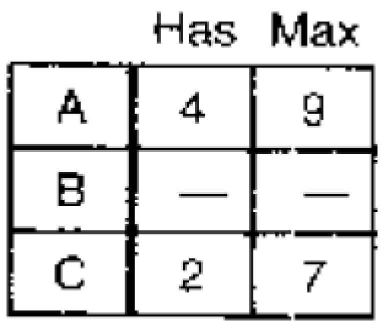
(b)



	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)



	Has	Max
A	4	9
B	—	—
C	2	7

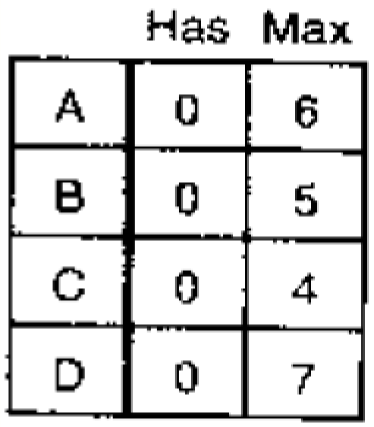
Free: 4

(d)

Demonstration that the state in (b) is not safe

Banker's Algorithm for a Single Resource

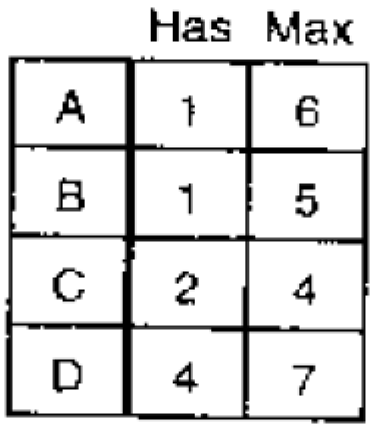
Available unit: 10; Requested: 22
How requested resources are allocated?



	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

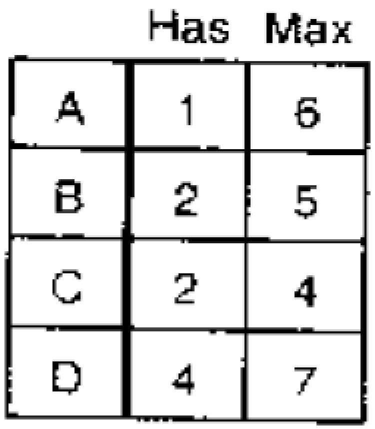
(a)



	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)



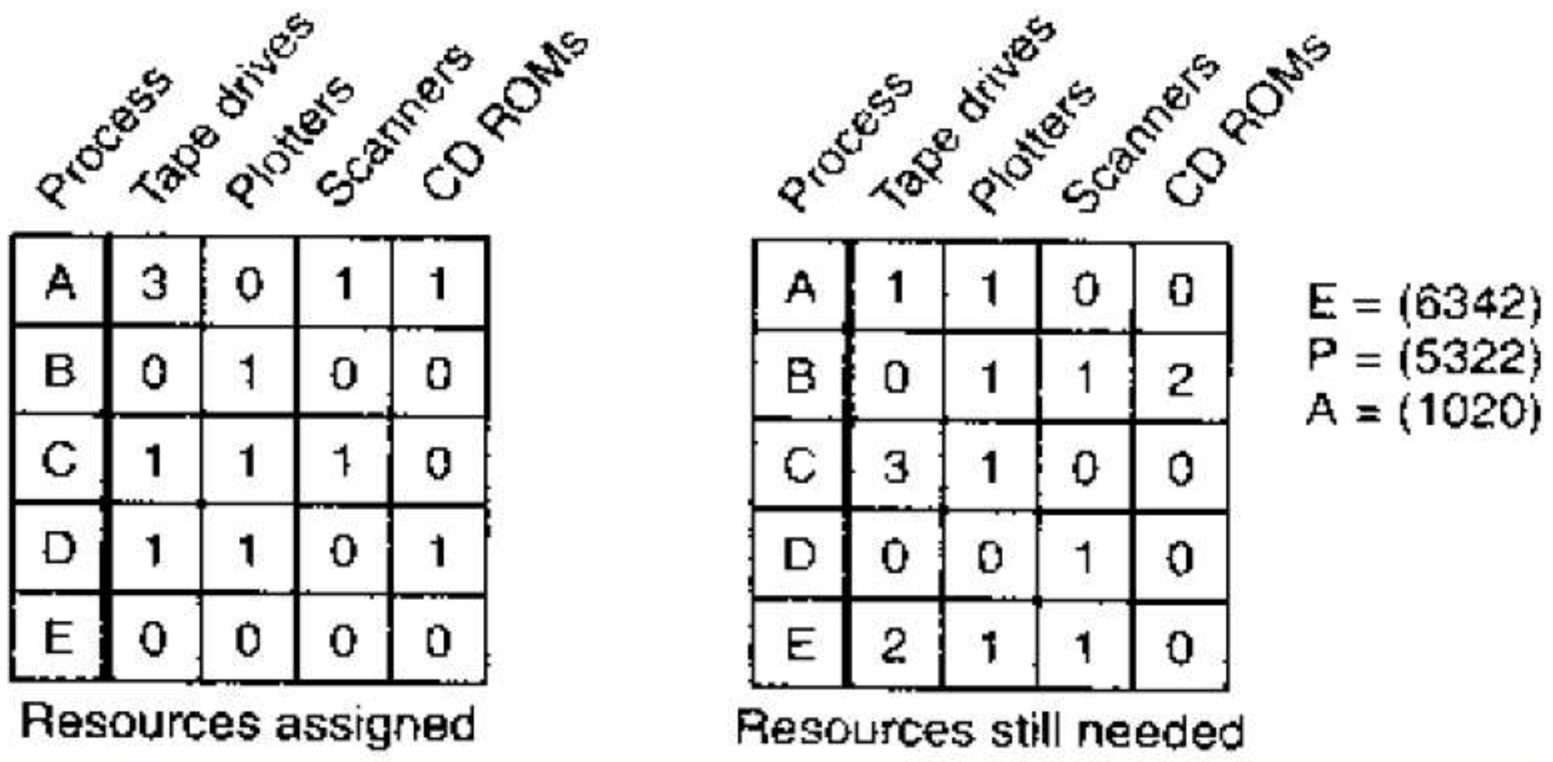
	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

Banker's Algorithm for Multiple Resources



Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

E: Existing Resources(6 tape drivers, 3 plotters, 4 scanners & 2 CD ROMs)
P: Possessed Resources(5 tape drivers, 3 plotters, 2 scanners & 2 CD ROMs)
A: Available Resources(1 tape drivers, 0 plotters, 2 scanners & 0 CD ROMs)
Process of execution: <D, A, E, B, C>

Algorithm

- The algorithm for checking to see if a state is safe:
  - 1. Look for row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.
  - 2. Assume the process of the row chosen requests all the resources it needs(which is granted to be possible) and finishes. Mark that process as terminated and add all its resource to the A vector
  - 3. Repeat steps 1 and 2 until either all processes are marked terminated, in which case the initial state was safe, or until a deadlock occurs, in which case it was not.

Deadlock Prevention

- Deadlock avoidance is difficult, because it requires information about feature requests, which is not known.
- Prevent occurrence of deadlock by ensuring that at least one of the four necessary conditions for deadlock cannot hold

1. Attacking the Mutual Exclusion Condition
   - Principle:
     - avoid assigning resource when not absolutely necessary. Let the process to use resources simultaneously
   - Impose some condition on processes
     - **E.g. Spooling** – By spooling printer output, several processes can generate output at the same time but the only process that actually requests the physical printer is the printer daemon.

- **Disadvantage**: This solution cannot be used for every resource. E.g. a mutex lock cannot be simultaneously shared by several processes

Deadlock Prevention(cont'd)

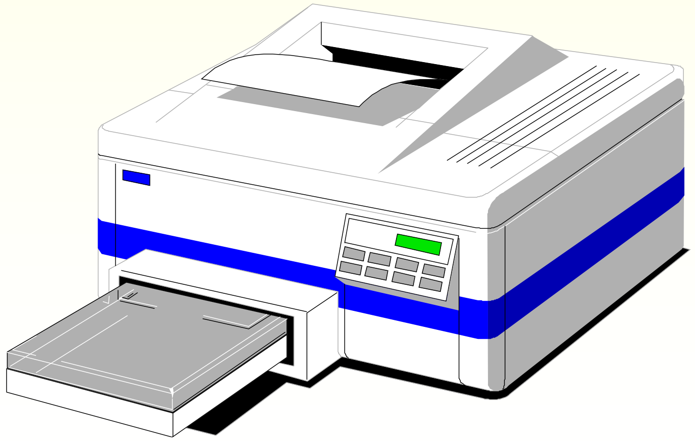
2. Attacking the Hold and Wait Condition

- Require processes to request all resources before starting
  - A process never has to wait for what it needs
- Problems:
  - Most processes do not know that resources they need before running
  - Resources will not be used optimally
- Variation:
  - process must give up all resources
  - then request all at once

Deadlock Prevention(cont'd)

3. Attacking the No Preemption Condition

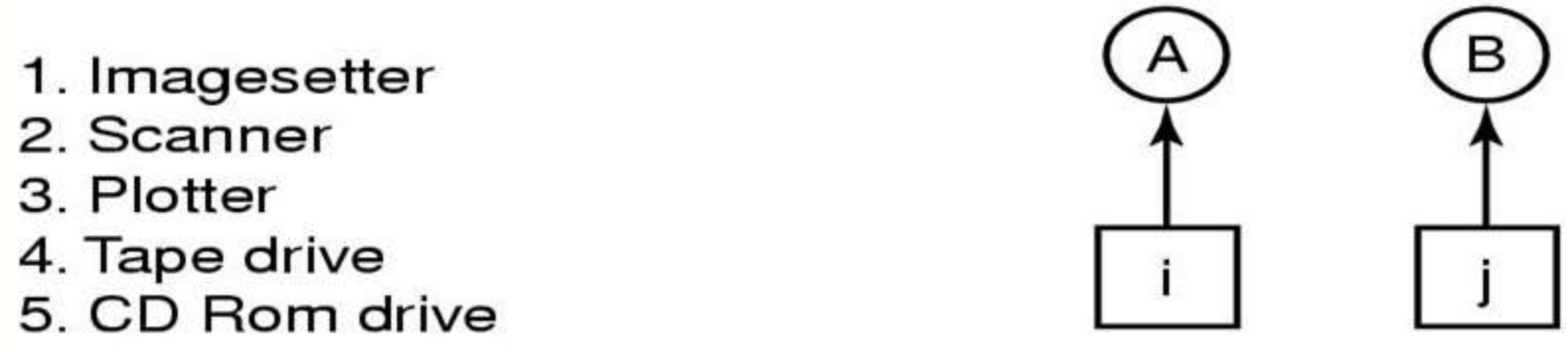
- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- This is not a feasible option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer because a needed plotter is not available
  - !!??



Deadlock Prevention(cont'd)

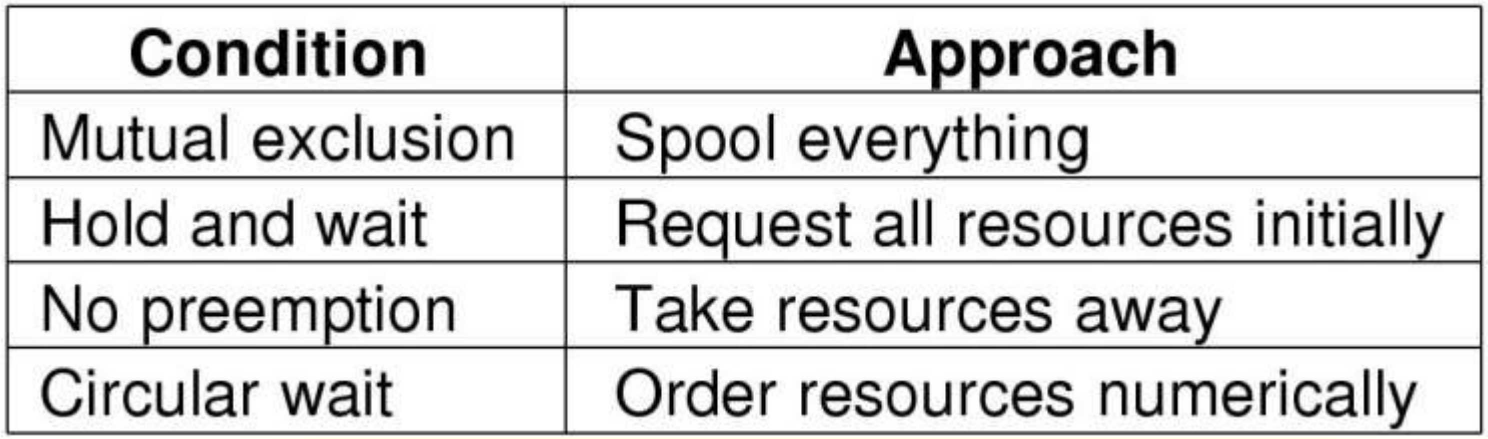
4. Attacking the Circular Wait Condition

- Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration
  - In other words, in order to request resource Rj, a process must first release all Ri such that i <= j.
- One big challenge in this scheme is determining the relative ordering of the different resources



1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

Summary



Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Summary of approaches to deadlock prevention