Dt282 Operating Systems 2

Deadlock and Starvation

Introduction

- Resource sharing perspectives
  - Memory management and processor sharing
- Many programs competing for limited resources
- Lack of process synchronization consequences
  - Deadlock: “deadly embrace,” “Catch 22,” “blue screen of death,” etc.
    - Two or more jobs placed in HOLD state
    - Jobs waiting for unavailable vital resource
    - System comes to standstill
    - Unresolved by OS: requires external intervention

Deadlock, Livelock, and Starvation

- Deadlock:
  - Two processes are unable to proceed because *each* is waiting for the other to “do” something.

- Livelock:
  - Two or more processes continually change their state without doing any useful work (e.g. disk sharing)

- Starvation:
  - One of the processes is continually rolled back or selected as victim for recovery from deadlock

Deadlock

- More serious than starvation
- Affects entire system
  - Affects more than one job
  - All system resources become unavailable
- More prevalent in interactive systems
- Real-time systems
  - Deadlocks quickly become critical situations
- OS must prevent or resolve deadlock

Cases of Deadlock

- Non-sharable/non-preemptable resources
  - Files, printers or scanners
  - Allocated to jobs requiring same type of resources (sharable resources like database and disks…)
- Resource types locked by competing jobs
  1. File requests
  2. Databases
  3. Multiple device allocation
  4. Disk sharing

The race problem

- Race (lost update problem) between processes
  - Results when locking not used and no concurrency enforcement policy
  - Causes incorrect final version of data
  - Depends on process execution order
  - Prevented by using concurrency locking (refer to lecture mutexs: test-set, wait and signal and semaphores)

Non Synchronisation example

- if two processes or threads/transactions try to update the same object can result in: **Race/lost update problem**

- Consider v = v + 3:
  1. Thread A fetches *v* = 5 into a processor register within processor A
  2. Thread B fetches *v* = 5 into a processor register within processor B
  3. Thread B increments value in its processor register to 9
  4. Thread B stores 9 into *v*
  5. Thread A increments value in its processor register to 9
  6. Thread A stores 9 into *v*

A way of modeling Deadlocks

- Directed graphs: Richard Holt (1972)
  - Circles represent processes
  - Squares represent resources
  - Line with arrow from resource to process
    - Process holding resource
  - Line with arrow from a process to resource
    - Process waiting for resource
  - Arrow direction indicates flow
  - Cycle in graph
    - Deadlock involving processes and resources

Case 1: Deadlocks on File Requests

- Jobs request and hold files for execution duration
- Example (Figure 5.7)
  - Two programs (P1, P2) and two files (F1, F2)
  - Deadlock sequence
    - P1 has access to F1 and also requires F2
    - P2 has access to F2 and also requires F1
  - Deadlock remains until:
    - One program is closed *or*
    - One program is forcibly removed and file is released
  - Other programs requiring F1 or F2
    - Put on hold for duration of situation

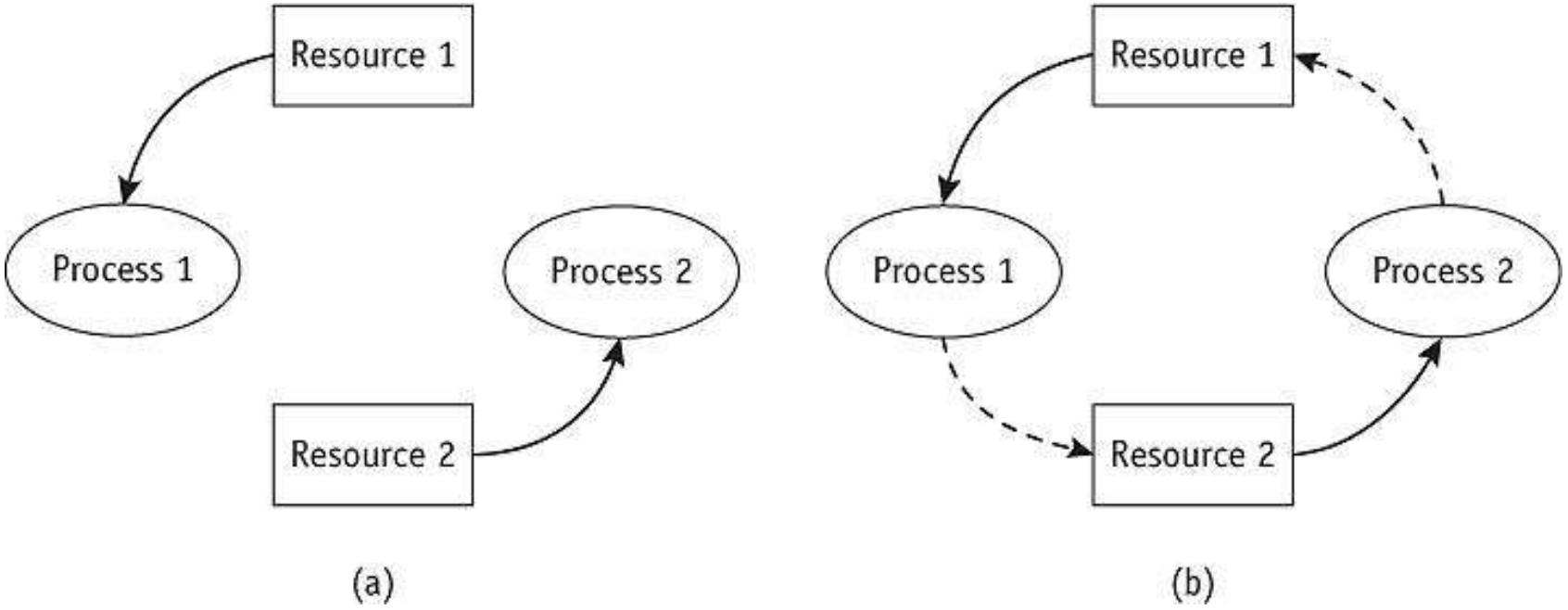
Necessary Conditions for Deadlock or Livelock

- **Four conditions** required for a locked system
  1. **Mutual exclusion**: allowing only one process access to dedicated resource
  2. **Resource holding and waiting**: may hold a resource (not releasing the resource); while waiting assignment of another resource, held by another job (for other job to retreat)
  3. **No pre-emption**: no resource can be forcibly removed from a process holding it
  4. **Circular wait**: each process waiting for another to voluntarily release so at least one can continue

- **All conditions** required for deadlock

- Resolving deadlock: **remove one** of the conditions

Modeling Deadlocks (cont'd.)



(figure 5.7)

1. In (a), Resource 1 is being held by Process 1 and Resource 2 is held by Process 2 in a system that is not deadlocked.
2. In (b), Process 1 requests Resource 2 but doesn't release Resource 1, and Process 2 does the same— creating a deadlock. (If one process released its resource, the deadlock would be resolved.)

Modeling Deadlocks (cont'd.)

- In the following examples there are:
  - System has three processes (P1, P2, P3)
  - System has three resources (R1, R2, R3)
- Three graph scenarios to help detect deadlocks

1. Scenario one: no deadlock
   - Resources released before next process request
2. Scenario two: deadlock
   - Processes waiting for resource held by another
3. Scenario three: no deadlock
   - Resources released before deadlock

Modeling Deadlocks (cont'd.)

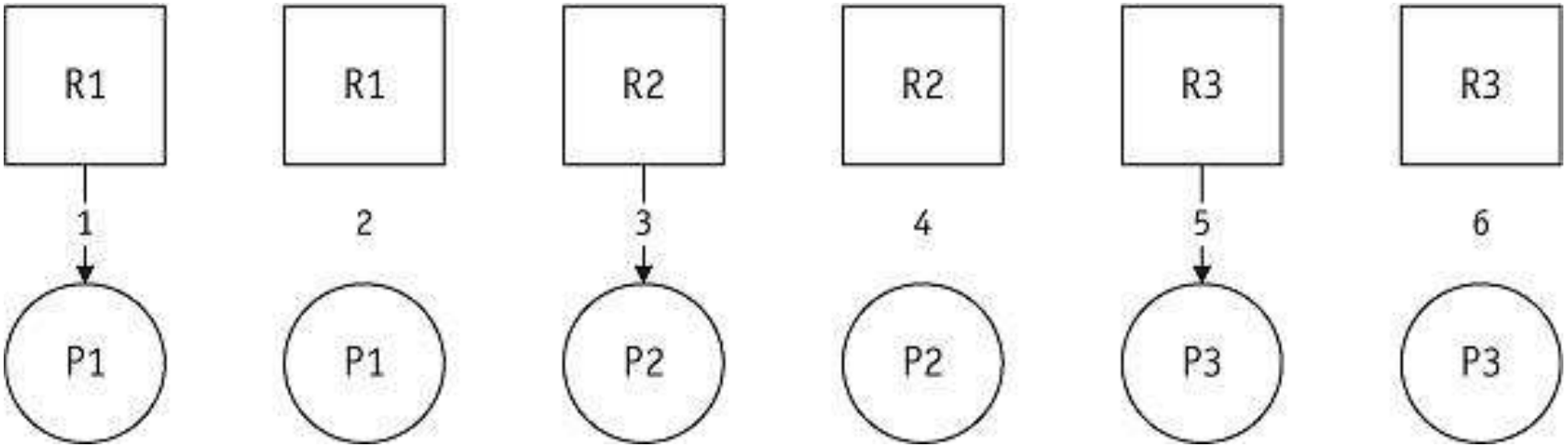
- **No deadlock**
  - Resources released before next process request

Event	Action
1	Process 1 (P1) requests and is allocated the printer (R1).
2	Process 1 releases the printer.
3	Process 2 (P2) requests and is allocated the disk drive (R2).
4	Process 2 releases the disk drive.
5	Process 3 (P3) requests and is allocated the plotter (R3).
6	Process 3 releases the plotter.

(table 5.1)
Scenario 1. These events are shown in the directed graph in Figure 5.8.

Modeling Deadlocks (cont'd.)

- **No deadlock**
  - Resources released before next process request



(figure 5.8)
First scenario. The system will stay free of deadlocks if *each resource is released before it is requested by the next process*.

Modeling Deadlocks (cont'd.)

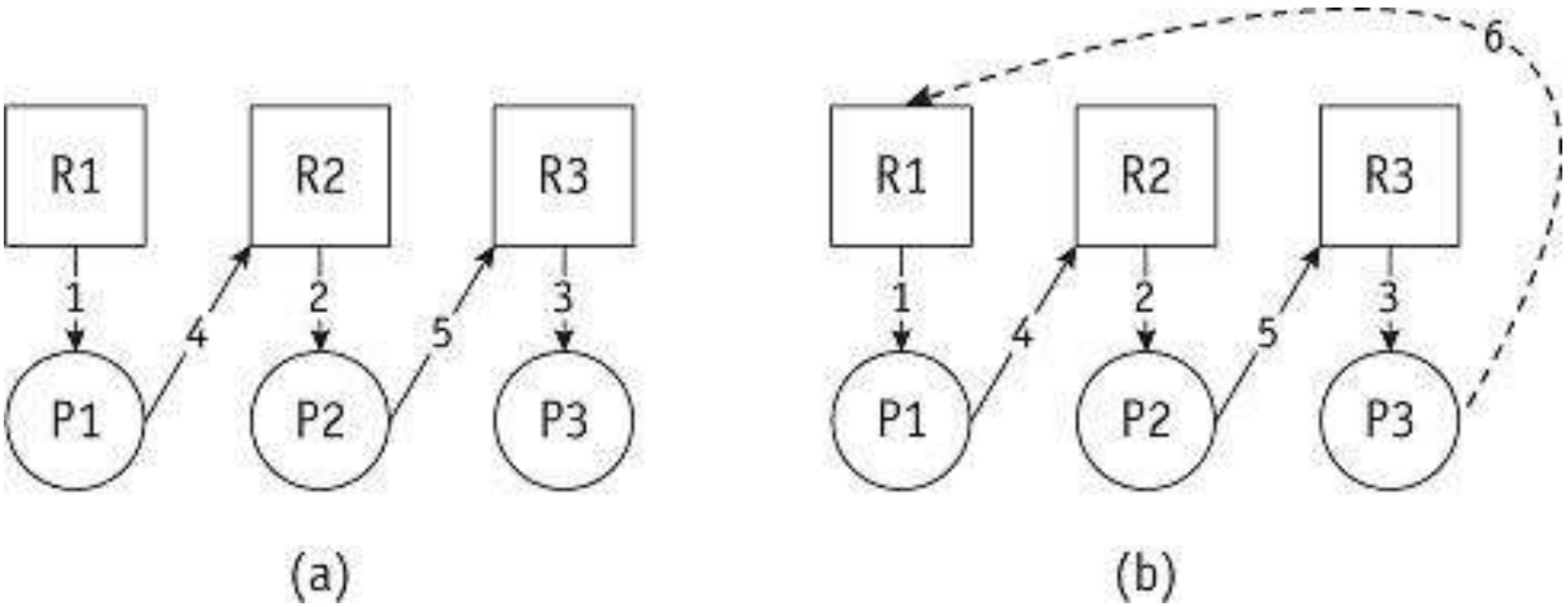
- Deadlock
  - Processes waiting for resource held by another

(table 5.2) Scenario 2. This sequence of events is shown in the two directed graphs shown in Figure 5.9.

Event	Action
1	P_1 requests and is allocated R_1.
2	P_2 requests and is allocated R_2.
3	P_3 requests and is allocated R_3.
4	P_1 requests R_2.
5	P_2 requests R_3.
6	P_3 requests R_1.

In Event 4-6 each request is blocked (possibly leading to deadlock)

Modeling Deadlocks (cont'd.)



(figure 5.9)
Second scenario. The system (a) becomes deadlocked (b) when P3 requests R1. as represented by the circular wait shown here by dashed line from P3 to R1. In this case all the 4 conditions for deadlock are fulfilled.

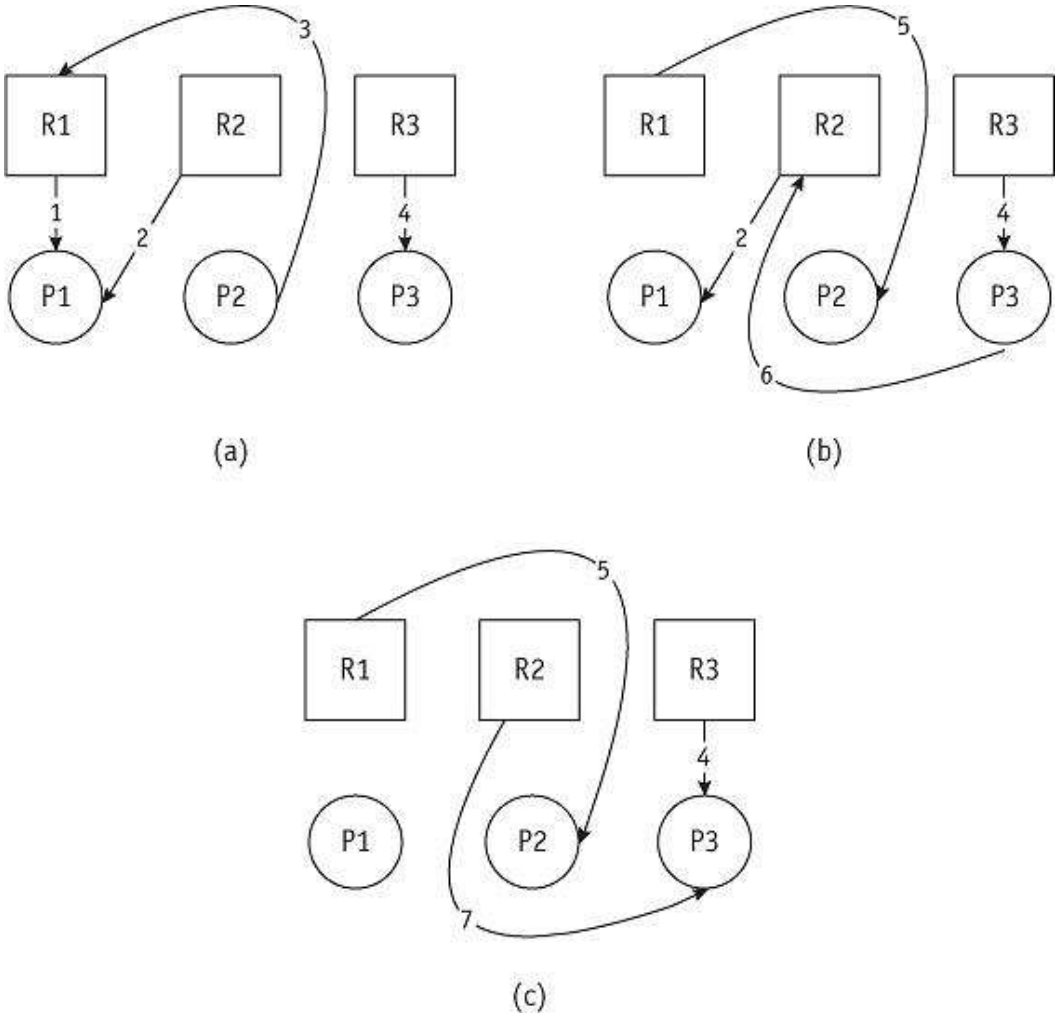
Modeling Deadlocks (cont'd.)

- No deadlock
  - Resources released before deadlock

Event	Action
1	P1 requests and is allocated R1.
2	P1 requests and is allocated R2.
3	P2 requests R1.
4	P3 requests and is allocated R3.
5	P1 releases R1, which is allocated to P2.
6	P3 requests R2.
7	P1 releases R2, which is allocated to P3.

(table 5.3)
Scenario 3. This sequence of events is shown in the directed graph in Figure 5.10.

Modeling Deadlocks (cont'd.)



(figure 5.10)
The third scenario. After event 4, the directed graph looks like (a) and **P2 is blocked** because P1 is holding on to R1. However, event 5 breaks the deadlock and the graph soon looks like (b). Again there is a blocked process, P3, which must wait for the release of R2 in event 7 when the graph looks like (c).

Refer to table 5.3 for 1…7

Three Strategies for Handling Deadlocks

- Prevention
  - Prevent occurrence of one condition
    - Mutual exclusion, resource holding and waiting, no pre-emption, circular wait

1. Avoidance
   - Avoid deadlock if it becomes “probable”

2. Detection and Recovery
   - Detect deadlock when it occurs
   - Resume system normally, quickly and gracefully

Deadlock Prevention

- **Prevention eliminates any of the *four* conditions**
  - Mutual exclusion
    - Some resources must be allocate exclusively so can not always be prevented.
    - However some resources, mutual exclusion, can be bypassed if I/O device uses spooling (temporary store data) (may cause spooling deadlock)
  - Resource holding
    - Bypassed if jobs request every necessary resource at creation time; e.g. batch systems in advance given all required memory. (no memory resource deadlock)
    - However, the degree of Multiprogramming (memory allocation) decrease significantly and have Idle peripheral devices that are allocated to jobs (that may not use them all the time)

Deadlock Prevention

- Prevention (cont'd.)
  - No preemption
    - Bypassed if operating system allowed to deallocate resources from jobs
    - Okay if job state easily saved and restored (round robin algorithm or page swapping)
    - Pre-empting dedicated I/O device or files during modification requires recovery tasks: undo/redo…
  - Circular wait
    - Requires jobs to anticipate resource request order of resources
    - *One approach the resources must be requested using specific ordering schemes (printer = 1, disk =2…*
    - So if the actual resource order was disk and printer it is still required to request printer first even thought it is will be used later…
    - Difficult to satisfy all users

Deadlock: Avoidance

- Avoidance: checks if the “new” state is safe or unsafe
  - System knows ahead of time
    - Sequence of requests associated with each active process
- Dijkstra’s Bankers Algorithm (Dijkstra, 1965)
  - Regulates resource allocation to avoid deadlocks
    - No customer granted loan exceeding bank’s total capital
    - All customers given maximum credit limit
    - No customer allowed to borrow over limit
    - Sum of all loans will not exceed bank’s total capital

Deadlock: Avoidance

- Operating systems deadlock avoidance assurances
  - Never satisfy request if job state moves from safe to unsafe
    1. Begin by Identify job with *smallest number* of remaining resources to complete the job
    2. Number of available resources >= number needed for selected job to complete: safe state
    3. A job is block whose request is jeopardizing safe state or *unsafe state*
    4. *Repeat this process for the remaining jobs if it is in a safe state.*

Strategies for Handling Deadlocks (cont'd.)

Customer	Loan Amount	Maximum Credit	Remaining Credit
Customer #1	0	4,000	4,000
Customer #2	2,000	5,000	3,000
Customer #3	4,000	8,000	4,000
Total loaned: $6,000			
Total capital fund: $10,000			

(table 5.4)
The bank started with $10,000 and has remaining capital of $4,000 after these loans. Therefore, it's in a "safe state."

Strategies for Handling Deadlocks (cont'd.)

Customer	Loan Amount	Maximum Credit	Remaining Credit
Customer #1	2,000	4,000	2,000
Customer #2	3,000	5,000	2,000
Customer #3	4,000	8,000	4,000
Total loaned: $9,000			
Total capital fund: $10,000			

(table 5.5)
The bank only has remaining capital of $1,000 after these loans and therefore is in an “unsafe state.”

Deadlock: Avoidance (safe state)

Job No.	Devices Allocated	Maximum Required	Remaining Needs
Job 1	0	4	4
Job 2	2	5	3
Job 3	4	8	4
Total number of devices allocated: 6			
Total number of devices in system: 10			

(table 5.6)
Resource assignments after initial allocations. This is a safe state: Six devices are allocated and four units are still available. *Note: this is only showing **one** resource.*

Deadlock: Avoidance (unsafe state)

Job No.	Devices Allocated	Maximum Required	Remaining Needs
Job 1	2	4	2
Job 2	3	5	2
Job 3	4	8	4
Total number of devices allocated: 9			
Total number of devices in system: 10			

(table 5.7)
Resource assignments after later allocations. This is an unsafe state: Only one unit is available but every job requires at least two to complete its execution.

Class Exercise: Bankers algorithm

Multiple devices

- What is the contents of the need matrix:
- Have any process their resource requests satisfied?
- Is the system safe and if so what is the safe process sequence?
- P_o no P_1 yes…. Available = Allocation + available ;
- What is the allocated after p is finished.

Process	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	

Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

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

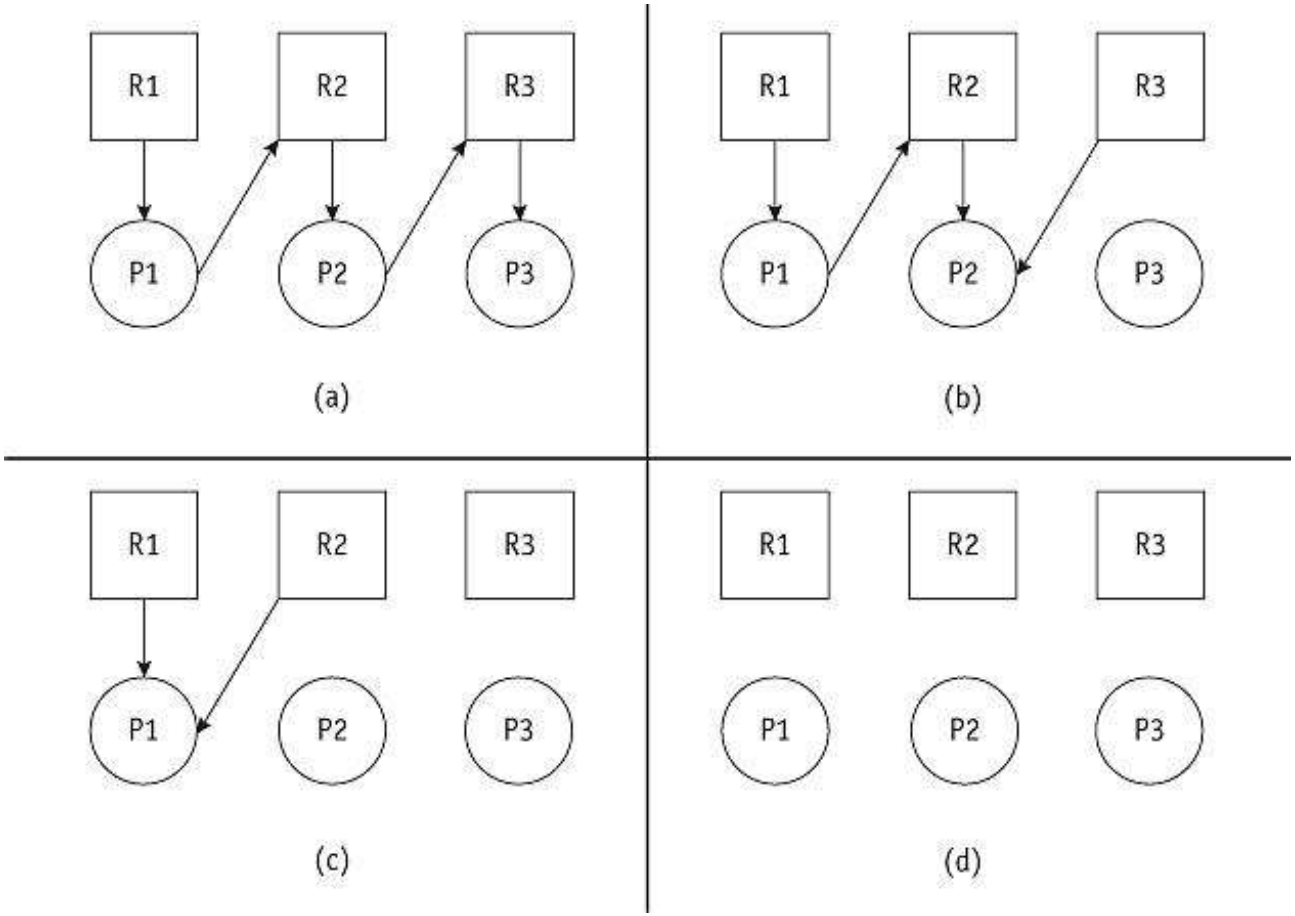
Deadlock: Avoidance

- Problems with the Banker's Algorithm
  - Jobs must state maximum number needed resources
  - Requires constant number of total resources for each class (problem is a resource breaks)
  - Possible high overhead cost incurred
  - Resources not well utilized
    - Algorithm assumes worst case
  - Scheduling suffers
    - Result of poor utilization
    - Jobs kept waiting for resource allocation

Deadlocks: Detection and Recovery

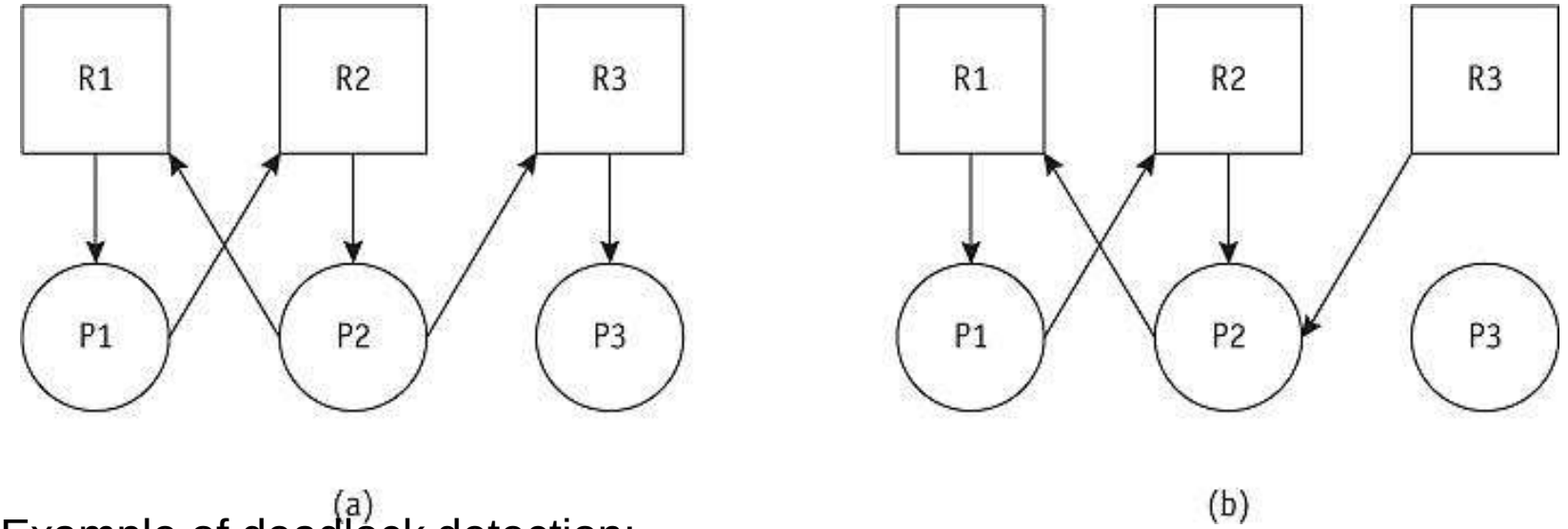
- **Detection**: build directed resource graphs
  - Look for, potential, cycles
- Algorithm detecting circularity
  - Executed whenever appropriate; at specific times or system performance deteriorates
- Detection algorithm and steps in reducing them
  1. Remove process using current resource and not waiting for one fig 11.b P3
  2. Remove process that have all resource requests allocated ; e.g. P2 in fig 11.c
  3. Go back to step 1
     - Repeat steps 1 and 2 until all connecting lines removed

Detection and Recovery: Graph Reduction



(figure 5.11) This system is deadlock-free because the graph can be completely reduced, as shown in (d).

Detection and Recovery: Graph Reduction



Example of deadlock detection:

Even after this graph is reduced as much as possible (by removing the request from P3), a -> b
it is still deadlocked as b can not be reduced any further

Deadlock: Detection and Recovery

- Recovery
  - Deadlock must be untangled once detected
  - System returns to normal quickly
- All recovery methods have at least one victim
- Recovery methods
  1. Terminate every job active in system
     - Restart jobs from beginning
  2. Only Terminate all jobs involved in deadlock
     - Ask users to resubmit jobs
  3. Identify jobs involved in deadlock
     - Terminate “deadlocked” jobs *one at a time* and see if it resolves deadlock

Deadlock: detection and recovery

- Factors to consider
  - Select victim with least-negative effect on the system
  - Most common
    - Job priority under consideration: high-priority jobs usually untouched
    - CPU time used by job: jobs close to completion usually left alone
    - Number of other jobs affected if job selected as victim
    - Jobs modifying data: usually not selected for termination (a database issue)

Starvation

- Job execution prevented
  - Waiting for resources that “may never” become available.
  - Results from conservative resource allocation in avoidance or possibly selecting same victim each time

- The starvation concept is illustrate in the dining philosophers problem (if interested a link to detail and sample program can be found here:)

- Starvation avoidance
  - Implement algorithm tracking how long each job has been waiting for resources (aging)
  - Increasing priority of a process as it ages and blocks new jobs until completed

Sample Exam Questions

- What the four conditions necessary for *Deadlock* to occur? **(4 Marks)**

- Explain, using conventional deadlock modelling, if the following schedule result in deadlock: **(6 marks)**

Event	Action
1	P1 requests and is allocated R1.
2	P2 requests and is allocated R2.
3	P3 requests and is allocated R3.
4	P1 requests R2.
5	P2 requests R3.
6	P3 requests R1.

- What are the three ways of preventing deadlock? **(4 marks)**

Sample question Deadlock detection and graph reduction

Figure: A

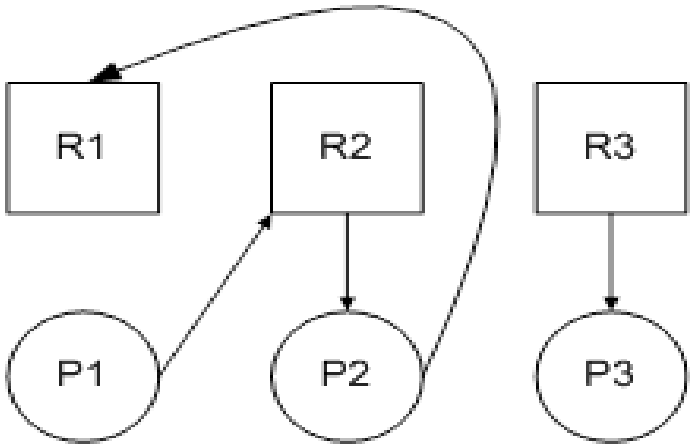


Figure: B

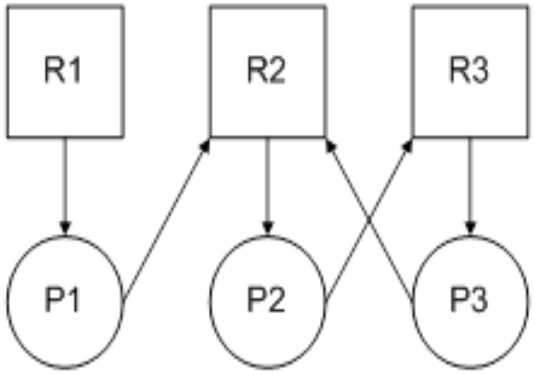


Fig A and Fig B show directed resource graphs for three processes (P1, P2 and P3 and three sources (R1, R2, R3). Using the detection/reduction algorithm Explain: For each of the above diagrams show: **(14 marks)**

If Is it deadlocked?

What is the result after reduction?

Which, if any processes are deadlocked

Sample question (bankers algorithm)

	Allocation	Max	Need	Available
				3 2 2 1
	A B C D	A B C D	A B C D	A B C D
P0	4 0 0 1	7 0 2 1		
P1	1 1 0 0	1 6 5 0		
P2	1 0 4 5	3 3 4 6		
P3	0 4 2 1	1 5 6 2		
P4	0 3 1 2	2 4 3 2		

Using Bankers algorithm answer the following:

1. **How many resources of type A, B, C and D are there?**
2. **What are the contents of the Need matrix?**
3. **Is the system in a safe state? Provide reasoning for your answer (show the sequence in which the processes would finish)**
4. **If a request from process P2 arrives for additional resources of {0, 2, 0, 0}, can the Bankers algorithm grant the request immediately? Provide reasoning for your answer. (14 Marks)**

Sample Question

- What is the relationship between deadlock and starvation. Give two reasons why starvation can occur and how it can be resolved. (4 marks)