Chapter 7: Dead Locks

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.

Introduction

- A system consists of a finite number of resources to be distributed among a number of competing processes.
- The resources are partitioned into several types, each consisting of some number of identical instances.
- Memory space, CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples

System Model

- A process must request a resource before using it and must release the resource after using it.
- A process may request as many resources as it requires to carry out its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system.

System Model

- Under the normal mode of operation, a process may utilize a resource in only the following sequence:
- 1. Request. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- 2. Use, The process can operate on the
- 3. Release. The process releases the resource.

System Model

- A system table records whether each resource is free or allocated; for each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

System Model

- A set of processes 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.
- The events with which we are mainly concerned here are resource acquisition and release.

Definition

- A deadlock state occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
- In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Deadlock Characterization

What are the features necessary for deadlocks to happen?

- A deadlock situation can arise if the following four conditions hold simultaneously in a system:

Mutual exclusion.

- At least one resource must be held in a non-sharable mode;
- That is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

Hold and wait.

- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

No preemption.

- Resources cannot be preempted.; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait

- A set {P_0, P_i, ..., P_n *of waiting processes must exist such* that *P-$_0$ is waiting for a resource held by* P_1, P_1 *is waiting for a resource* held by P2, •••, an so on.

- All four conditions must hold for a deadlock to occur.

Resource-Allocation Graph

- Deadlocks can be described more precisely in terms of a directed graph called a **system resource-allocation graph.**
- This graph consists of a set of vertices *V* and a set of edges *E*.
- *The set of vertices V is partitioned into two different types* of nodes:
- P - *{P_i, P_2,,.., P_n, the set consisting of all the active processes in the* system,
- *R = {R[, R_2, •••/ R_m}, the set consisting of all resource types in the* system.

Resource-Allocation Graph

- A directed edge from process P_i *to resource type* R_j *is denoted by* P_i -> R_j . it signifies that process P, has requested an instance of resource type R, and is currently waiting for that resource.

- A directed edge from resource type R_j to process P_i *is denoted by* R_j -» P_i . *it signifies that an instance of resource* type R_j *has been allocated to process* P_i

Resource-Allocation Graph

- A directed edge P_i -> R_j *is called a* **request edge; a directed edge** R_j -» P_i ***is called an assignment edge.***
- Pictorially, we represent each process P_i as a circle and each resource type R_j as a rectangle.
- Since resource type R_j may have more than one instance, we represent each such instance as a dot within the rectangle.

Resource-Allocation Graph Example

Consider the following

- The sets P, *R, and £:*
- $P = \{P_1, P_2, P_3\}$
- *R= {R1, R2,R3, R4}*
- £ = {P_1-> R_1, P2 _> R3, R_1 _> p2, R2 _> P2, R2 _> p1, R3 -> P3 }

Resource-Allocation Graph Example

Consider the following

- Resource instances:
- One instance of resource type R1
- *Two instances of resource type* R2
- One instance of resource type R3
- *Three instances of resource type* R4

Resource-Allocation Graph Deadlock Example

Suppose

- Suppose that process P3 requests an instance of resource type *R2. Since no resource instance is currently available, a request edge P3* —>R2 is added to the graph
- Draw this graph…

Methods for Handling Deadlocks

Refresher Class Exercise

- The sets P, *R*, *and £:*
- $P = \{P_1, P_2, P_3\}$
- $R = \{R1, R2, R3\}$
- £ = $\{R_1 \text{-> } P_2,\ P_1 \text{ _> } R_1,\ R_2 \text{ _> } p_1,\ R_2 \text{ _> } P_2,\ R_3 \text{ _> } p_3,\ p_2 \text{ -> } R_3,\ p_3 \text{ -> } R_2\}$

- Using the information provided above, draw a resource allocation graph
- Comment on the resultant graph

Methods for Handling Deadlocks

- Deadlocks can be dealt with in three ways;

1. Use a protocol to prevent or avoid deadlocks, ensuring that the system will *never enter a deadlock state*.
2. Allow the system to enter a deadlock state, detect it, and recover.
3. Ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock Prevention

Deadlock Prevention

- As we discussed, for a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.
- How can we then can we prevent the conditions from holding?

Mutual Exclusion does not hold

- The mutual-exclusion condition must hold for non sharable resources. For example, a printer cannot be simultaneously shared by several processes.(intrinsically non sharable)
- Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock.
- we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non sharable

Hold and Wait does not hold

- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
- How can we achieve this?

Hold and Wait does not hold

- One protocol that can be used requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls.

- What are our thoughts on this approach?

Hold and Wait does not hold

- An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

- What are our thoughts on this approach?

Hold and Wait does not hold

- Consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.

No Preemption does not hold

- To ensure that this condition does not hold, we can use the following protocol.
- If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources currently being held are preempted.
- The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

No Preemption does not hold

- This protocol is often applied to resources whose state can be easily saved and restored later, such as CPU registers and memory space. It cannot generally be applied to such resources as printers and tape drives.

Circular Wait

- The circular wait, can be denied by imposing a total ordering on all of the resource types and than forcing, all processes to request the resources in order (increasing or decreasing)

Circular Wait

- For example, provide a global numbering of all the resources, as shown

  1 ≡ Card reader

  2 ≡ Printer

  3 ≡ Plotter

  4 ≡ Tape drive

  5 ≡ Card punch

Circular Wait

A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2).

Class Exercise

1. Describe how the circular wait will be avoided by imposing a total ordering on all of the resource types.
2. Using the refresher class exercise, describe how the deadlock that exists can be prevented by ensuring that either of the below conditions does not hold
   - ❖ Ensuring that non preemption does not hold
   - ❖ Ensuring that hold and wait does not hold
   - ❖ Ensuring that circular wait does not hold

Deadlock Avoidance

Deadlock Avoidance

- **Deadlock avoidance requires that the operating system be given in** advance additional information concerning which resources a process will request and use during its lifetime.
- With this additional knowledge, it can decide for each request whether or not the process should wait.
- Each request requires that in making this decision the system consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Deadlock Avoidance

- A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular wait condition can never exist.
- There are two deadlock avoidance algorithms implemented;
    - Resource allocation graph
    - Bankers algorithm

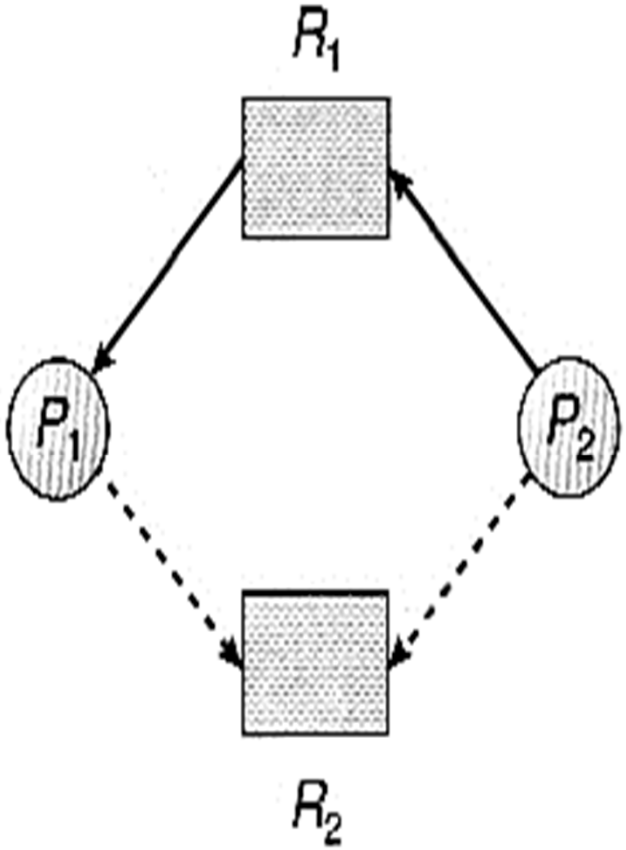
Resource-Allocation-Graph Algorithm

- This algorithm assumes one instance of a resource
- In addition to the request and assignment edges already described, we introduce a new type of edge, called a claim edge.

Resource-Allocation-Graph Algorithm

- A claim edge P …..> *Rj indicates that process P, may request resource R, at* some time in the future. This edge resembles a request edge in direction but is represented in the graph by a dashed line.
- When the process finally requests the resource, the claim edge is converted to a request edge.

Resource-Allocation-Graph for deadlock avoidance



Resource-Allocation-Graph Algorithm

- Can we allocate R2 to p2…

yes…Why?

No…Why?

Banker's Algorithm

- As discussed, The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type.
- The bankers algorithm is applicable to such a system but is less efficient than the resource-allocation graph scheme.
- The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

Banker's Algorithm

- By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Banker's Algorithm

- For the Banker's algorithm to work, it needs to know three things:
- How much of each resource each process could possibly request
- How much of each resource each process is currently holding
- How much of each resource the system currently has available

Banker's Algorithm

- Resources may be allocated to a process only if it satisfies the following conditions:
- request ≤ max, else set error condition as process has crossed maximum claim made by it.
- request ≤ available, else process waits until resources are available.
- Some of the resources that are tracked in real systems are memory, semaphores and interface access.

Deadlock Detection

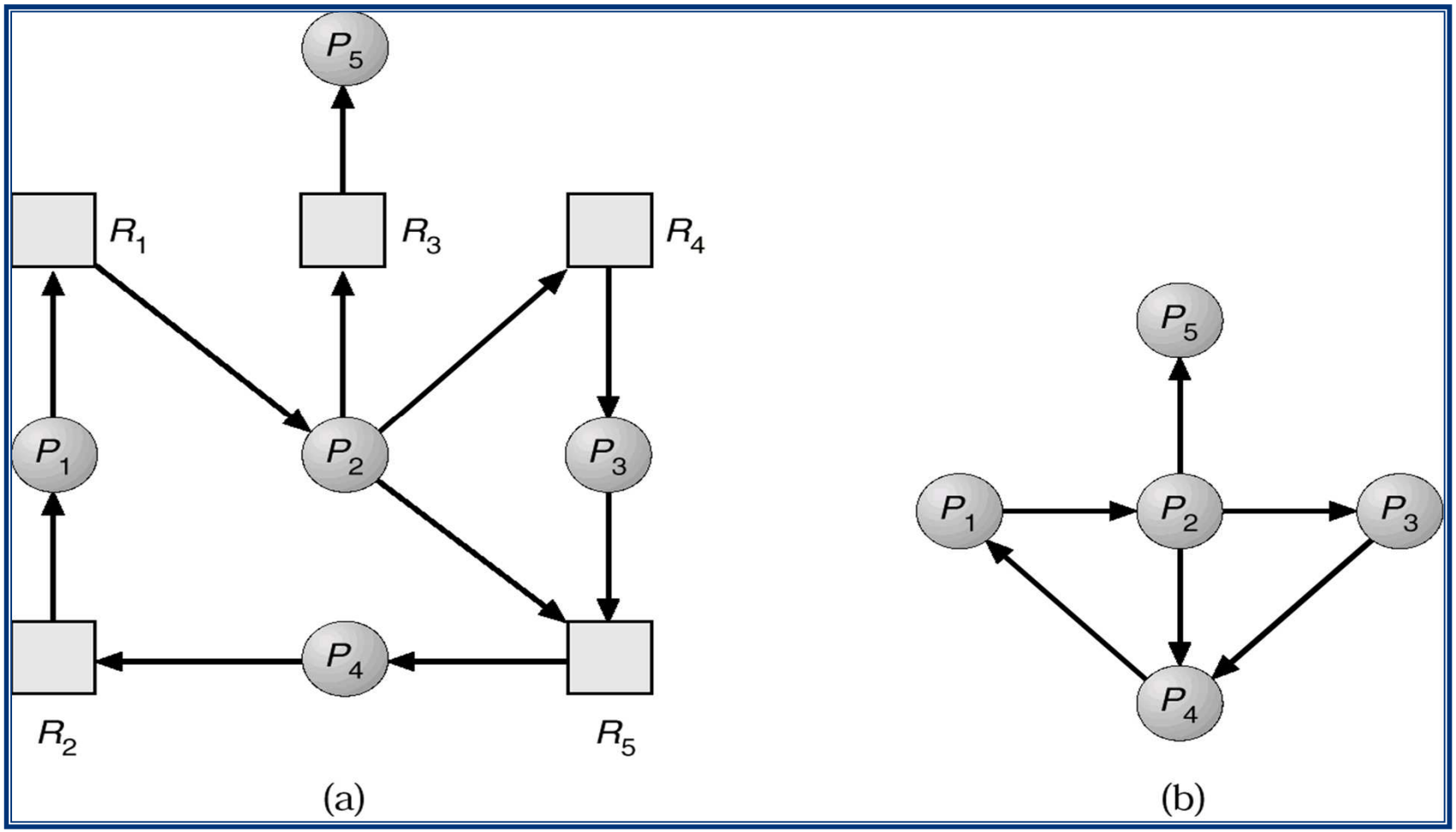
Deadlock detection

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system must provide:
  - An algorithm that examines the state of the system to determine whether a deadlock has occurred
  - An algorithm to recover from the deadlock

Deadlock detection

- If all resources have only a single instance, then we can define a deadlock-detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph.
- This graph is obtained from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.
- More precisely, an edge from p1 to p2 in a wait-for graph implies that process p1 is waiting for process p2 to release a resource that needs.

Resource-Allocation Graph & Corresponding wait-for graph



Deadlock detection

- To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.

Recovery From Deadlock

- When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system *recover from the deadlock automatically.*

Recovery From Deadlock

- *There* are two options for breaking a deadlock.
- One is simply to abort one or more processes to break the circular wait.
- The other is to preempt some resources from one or more of the deadlocked processes.

End of Chapter 7

Strathmore
UNIVERSITY

Ole Sangale Road, Madaraka Estate. PO Box 59857-00200, Nairobi, Kenya
Tel +254 (0)20 606155, 606268, 606380 Fax +254 (0)20 607498
Mobile +254 (0)722 25 428, (0)733 618 135 Email info@strathmore.edu
www.strathmore.edu