Module 6: Deadlock

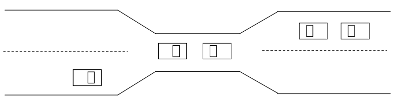
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example: System has 2 tape drives. P_1 and P_2 each hold one tape drive and each needs another one.
  Solution with 2 semaphores *A* and *B*, initialized to 1

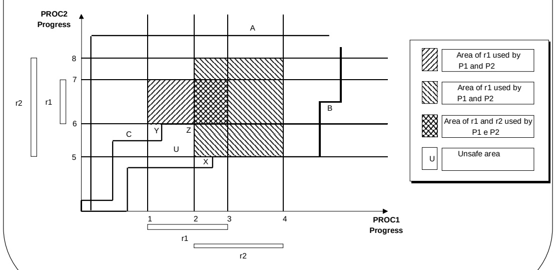
P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Bridge Crossing Example

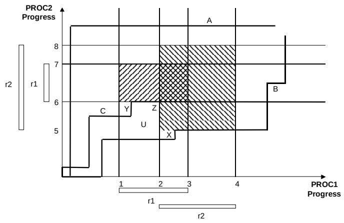


- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Joint progress of two processes



Methods of deadlock management



- Prevention: forbid the existence of an unsafe state
- Avoidance : forbid the entrance into an unsafe state
- Recovery: forbid the residence into an unsafe state

System Model

- Resource types R_1, R_2, . . ., R_m
  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource using the protocol:
  - request
  - use
  - release

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Resource-Allocation Graph

A set of vertices *V* and a set of edges *E*.

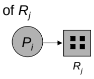
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

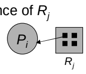
- Process

- Resource Type with 4 instances

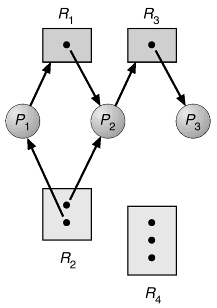
- P_i requests instance of R_j



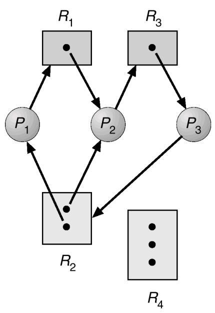
- P_i is holding an instance of R_j



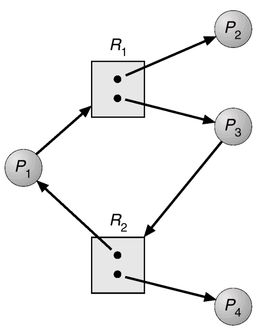
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock.
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

Restrain the ways request can be made.

Violate the conditions that are necessary for deadlock.

- Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources.

Deadlock Prevention

- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - ***Request All First*** (RAF): Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  - ***Release Before Request*** (RBR): process can request and wait for resources only if they don't keep resources already acquired. They must release their resources before they can require new resources.

Drawbacks:

Low resource utilization; starvation possible.

Deadlock Prevention

- **No Preemption**
  - 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 (preemption).
  - Preempted resources are added to the list of resources for which the process is waiting.
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Deadlock Prevention

- **Circular Wait** – *Hierarchical Resource Usage* (HRU).
- Impose a total ordering of all resource types, associating each resource to an integer number, for example:

  disks = 1

  tapes = 5

  printers = 12

- and require that each process requests resources in an increasing order of enumeration.

HRU

- A process can require an instance of resource r_k
  - if $F(r_k) > F(r_i)$, where r_i is a type of resource obtained before r_k.
- Otherwise, the process must release all the resources of type r_h such that $F(r_k) \leq F(r_h)$.

Operating Systems 6.19 Laface 2012 - Silberschatz and Galvin

Hierarchical Resource Usage

Let's suppose that there exists a set of processes $P = \{ P_0, P_1, \ldots, P_n \}$ in circular wait $\Rightarrow$

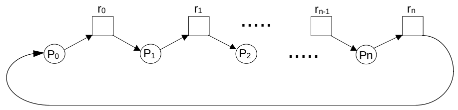
$$F(r_k) < F(r_{k+1}),\ \forall\ k = 0 .. n-1 \quad \text{i.e.}$$

$$F(r_0) < F(r_1) < \ldots < F(r_n) < F(r_0)$$

$$F(r_0) < F(r_0)$$

this is clearly absurd.

- A condition of circular wait, thus, cannot happen.



Operating Systems 6.20 Laface 2012 - Silberschatz and Galvin

Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

Operating Systems 6.21 Laface 2012 - Silberschatz and Galvin

Banker's Algorithm: State

Cash	2
P	4 (4)
Q	2 (1)
R	2 (7)

Acquired Resources (remaining requests)

Banker's Algorithm: Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a safe sequence of resource allocation for all processes.
- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

Banker's Algorithm

- Each process declare the *maximum number* of resources it will need
- When it requires a resource it can be blocked for a limited amount of time
- When it obtain the resources, it will release them in a finite amount of time

Complexity O(m x n^2), where m is the number of resources and n the number of processes.

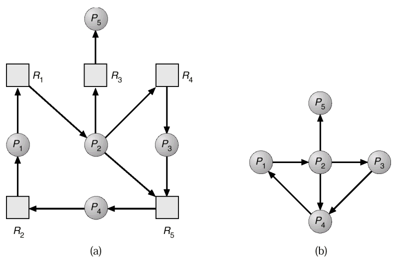
Example of safe state

Cash	2	Cash	4	Cash	8	Cash	10
P	4 (4)	P	4 (4)	P	- (-)	P	- (-)
Q	2 (1)	Q	- (-)	Q	- (-)	Q	- (-)
R	2 (7)	R	2 (7)	R	2 (7)	R	- (-)

Example of unsafe state

Cash	2	Cash	1	Cash	3
P	4 (4)	P	4 (4)	P	4 (4)
Q	2 (1)	Q	2 (1)	Q	- (-)
R	2 (7)	R	3 (6)	R	3 (6)

Resource-Allocation Graph And Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Deadlock Detection

- Allow system to enter deadlock state
- Maintain *wait-for* graph
  - Nodes are processes.
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
- Periodically, or when a process requires a resource, invoke an algorithm that searches for a cycle in the graph
- If a cycle exists the a recovery scheme is applied

An algorithm to detect a cycle in a graph requires

O(n^2) operations, where n is the number processes.

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?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

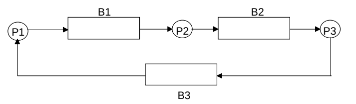
- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process fro that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Exampe of rollback scheme

- Threads that acquire 3 locks:
  - one thread in sequence 1, 2, and 3
  - the second thread in opposite sequence: 3, 2, and 1
- Typical condition of possible deadlock

backoff.c

Deadlock using IPC

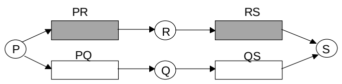


P1	P2	P3
...	...	...
send (B1,m)	send (B2,m)	send (B3,m)
...	...	...
send (B1,m)	send (B2,m)	send (B3,m)
...	...	...
receive (B3,m)	receive (B1,m)	receive (B2,m)

time

m = message
Bi = i-th communication buffer

Deadlock using IPC



P	Q	R	S
....	....	....	
send (PR,m)	....	send (RS,m)	....
....	....	....	....
send (PR,m)	receive (PQ,m)	send (RS,m)	receive (QS,m)
....			
send (PQ,m)	send (QS,m)	receive (PR,m)	receive (RS,m)