操作系统原理

第八章：死锁

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

The deadlock problem

The deadlock problem

- *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.*

The deadlock problem

- *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.*
- Example 1

The deadlock problem

- *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.*
- Example 1
  - The system has 2 tape drives.

The deadlock problem

- *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.*
- Example 1
  - The system has 2 tape drives.
  - P_1 and P_2 hold one tape drive and each needs another one.

The deadlock problem

- *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.*
- Example 1
  - The system has 2 tape drives.
  - P_1 and P_2 hold one tape drive and each needs another one.
- Example 2

The deadlock problem

- *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.*
- Example 1
  - The system has 2 tape drives.
  - P_1 and P_2 hold one tape drive and each needs another one.
- Example 2
  - Multi-threaded programs are good candidates for deadlock.

- *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.*
- Example 1
  - The system has 2 tape drives.
  - P_1 and P_2 hold one tape drive and each needs another one.
- Example 2
  - Multi-threaded programs are good candidates for deadlock.
  - Semaphores A and B, initialized to 1.

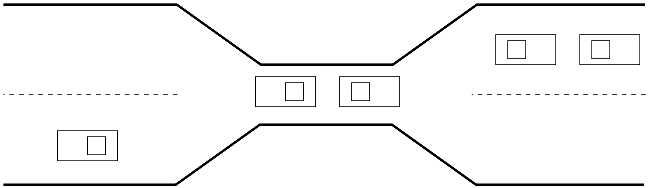
The deadlock problem

- *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.*
- Example 1
  - The system has 2 tape drives.
  - P_1 and P_2 hold one tape drive and each needs another one.
- Example 2
  - Multi-threaded programs are good candidates for deadlock.
  - Semaphores A and B, initialized to 1.

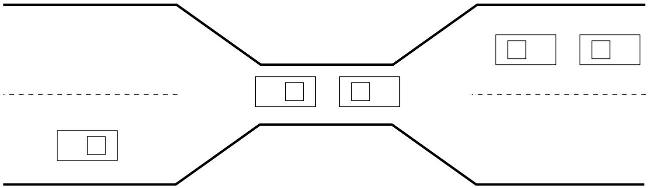
P_1	P_2
P(A);	P(B);
P(B);	P(A);

Bridge crossing example

Bridge crossing example

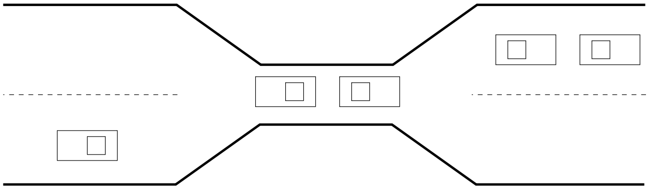


Bridge crossing example



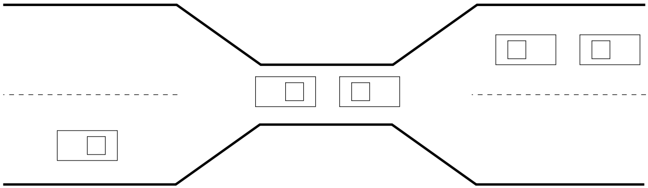
- Traffic only in one direction.

Bridge crossing example



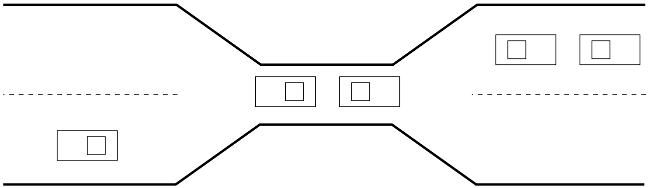
- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.

Bridge crossing example



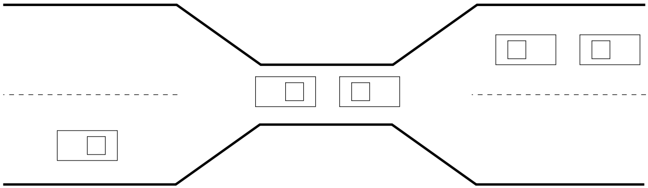
- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).

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.

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.

Outline

System model

System model

- A system consists of a finite number of resources to be distributed among a number of competing processes.

System model

- 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.

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request**

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request** - If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource;

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request** - If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource;
  2. **Use**

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request** - If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource;
  2. **Use** - The process is using the resource;

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request** - If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource;
  2. **Use** - The process is using the resource;
  3. **Release**

System model

- 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.
  - Resource types R_1, R_2, ..., R_m, such as CPU cycles, memory spaces and I/O devices.
- Each resource type R_i has W_i instances
- Actions to use a resource
  1. **Request** - If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource;
  2. **Use** - The process is using the resource;
  3. **Release** - The process releases the resources.

Deadlock characterization

Deadlock characterization

- Deadlock can arise if the following four conditions hold *simultaneously*.

Deadlock characterization

- Deadlock can arise if the following four conditions hold *simultaneously*.
  - **Mutual exclusion**

Deadlock characterization

- Deadlock can arise if the following four conditions hold *simultaneously*.
  - **Mutual exclusion** - only one process at a time can use a resource.

Deadlock characterization

- Deadlock can arise if the following four conditions hold *simultaneously*.
  - **Mutual exclusion** - only one process at a time can use a resource.
  - **Hold and wait**

Deadlock characterization

- Deadlock can arise if the following 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.

Deadlock characterization

- Deadlock can arise if the following 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**

Deadlock characterization

- Deadlock can arise if the following 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.

Deadlock characterization

- Deadlock can arise if the following 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**

Deadlock characterization

- Deadlock can arise if the following 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 $\{P_0, P_1, \ldots, P_n\}$ 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 (1/3)

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:
    1. $P = \{P_0, P_1, \ldots, P_n\}$, the set consisting of all the processes in the system;

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:
    1. $P=\{P_0, P_1, ..., P_n\}$, the set consisting of all the processes in the system;
    2. $R=\{R_0, R_1, ..., R_m\}$, the set consisting of all resource types in the system.

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:
    1. $P=\{P_0, P_1, ..., P_n\}$, the set consisting of all the processes in the system;
    2. $R=\{R_0, R_1, ..., R_m\}$, the set consisting of all resource types in the system.
  - Request edge

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:
    1. $P=\{P_0, P_1, \ldots, P_n\}$, the set consisting of all the processes in the system;
    2. $R=\{R_0, R_1, \ldots, R_m\}$, the set consisting of all resource types in the system.
  - Request edge - directed edge: $P_i \rightarrow R_j$;

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
    - V is partitioned into two types:
        1. $P=\{P_0, P_1, \ldots, P_n\}$, the set consisting of all the processes in the system;
        2. $R=\{R_0, R_1, \ldots, R_m\}$, the set consisting of all resource types in the system.
    - Request edge - directed edge: $P_i \rightarrow R_j$;
    - Assignment edge

Resource-allocation graph (1/3)

- Deadlocks can be described more precisely in terms of a *directed graph* called *resource-allocation graph*.
- This graph consists of a set of vertices V and a set of edges E.
  - V is partitioned into two types:
    1. $P = \{P_0, P_1, \ldots, P_n\}$, the set consisting of all the processes in the system;
    2. $R = \{R_0, R_1, \ldots, R_m\}$, the set consisting of all resource types in the system.
  - Request edge - directed edge: $P_i \rightarrow R_j$;
  - Assignment edge - directed edge: $R_j \rightarrow P_i$;

Resource-allocation graph (2/3)

Resource-allocation graph (2/3)

- A process is represented as a circle.

Resource-allocation graph (2/3)

- A process is represented as a circle.

Resource-allocation graph (2/3)

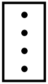
- A process is represented as a circle.

- A resource type with 4 instances.

Resource-allocation graph (2/3)

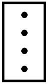
- A process is represented as a circle.

- A resource type with 4 instances.



- A process is represented as a circle.

- A resource type with 4 instances.

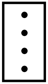


- P_i requests an instance of R_j.

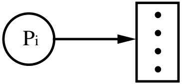
Resource-allocation graph (2/3)

- A process is represented as a circle.

- A resource type with 4 instances.



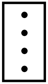
- P_i requests an instance of R_j.



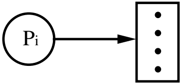
Resource-allocation graph (2/3)

- A process is represented as a circle.

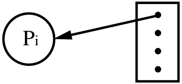
- A resource type with 4 instances.



- P_i requests an instance of R_j.



- P_i is holding an instance of R_j.

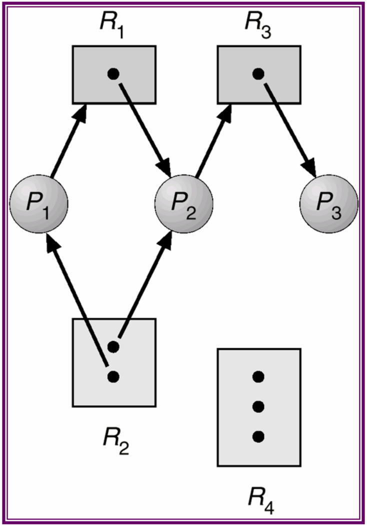


Resource-allocation graph (3/3)

Resource-allocation graph (3/3)

- An example

- An example



Deadlock and resource-allocation graph

Deadlock and resource-allocation graph

- Given the definition of a resource-allocation graph, it can be shown that

Deadlock and resource-allocation graph

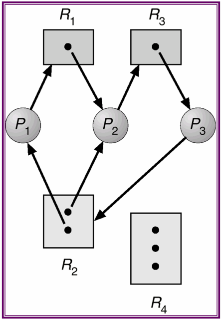
- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;

Deadlock and resource-allocation graph

- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;
  - If the graph does contain a cycle, then a deadlock *may* exist.

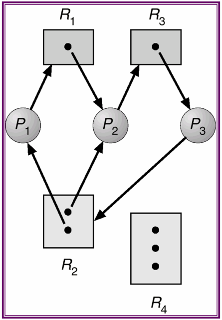
Deadlock and resource-allocation graph

- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;
  - If the graph does contain a cycle, then a deadlock *may* exist.



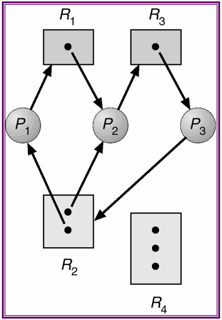
Deadlock and resource-allocation graph

- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;
  - If the graph does contain a cycle, then a deadlock *may* exist.

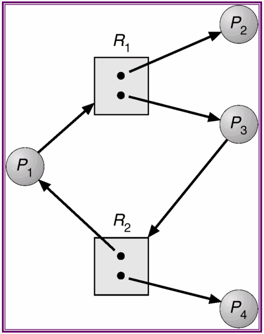


Deadlocked

- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;
  - If the graph does contain a cycle, then a deadlock *may* exist.

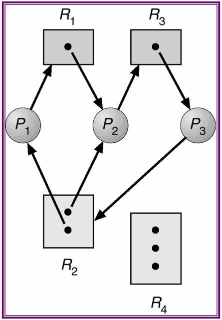


Deadlocked

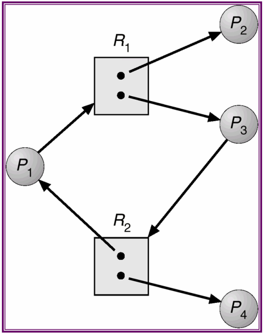


Deadlock and resource-allocation graph

- Given the definition of a resource-allocation graph, it can be shown that
  - If the graph contains no cycles, then no process in the system is deadlocked;
  - If the graph does contain a cycle, then a deadlock *may* exist.



Deadlocked



Not deadlocked

Outline

Methods of handling deadlock

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;
2. We can allow the system to enter a deadlock state, detect it, and recover from it;

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;
2. We can allow the system to enter a deadlock state, detect it, and recover from it;
3. We can ignore the problem altogether, and pretend that deadlocks never occur in the system.

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;
2. We can allow the system to enter a deadlock state, detect it, and recover from it;
3. We can ignore the problem altogether, and pretend that deadlocks never occur in the system.
    - This solution is used by *most* operating systems, including Windows and Unix.

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;
2. We can allow the system to enter a deadlock state, detect it, and recover from it;
3. We can ignore the problem altogether, and pretend that deadlocks never occur in the system.
    - This solution is used by *most* operating systems, including Windows and Unix.
    - Because the deadlocks occur very rarely and the deadlock-prevention, deadlock-avoidance or deadlock-detection and recovery algorithms are *costly*.

Methods of handling deadlock

1. We can design a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlock state;
2. We can allow the system to enter a deadlock state, detect it, and recover from it;
3. We can ignore the problem altogether, and pretend that deadlocks never occur in the system.
   - This solution is used by *most* operating systems, including Windows and Unix.
   - Because the deadlocks occur very rarely and the deadlock-prevention, deadlock-avoidance or deadlock-detection and recovery algorithms are *costly*.
   - It's a trade-off between *convenience* and *correctness*.

Questions

- Any questions?