Operating Systems

Lecture 6

Deadlocks

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Resources

- Anything in a computer system is a resource.
  - CPU
  - Memory
  - Disk space
  - Software resources: semaphore, file descriptor, …
  - …
- Some resources require exclusive access (CPU), some may be accessed by multiple processes at a time.

System model

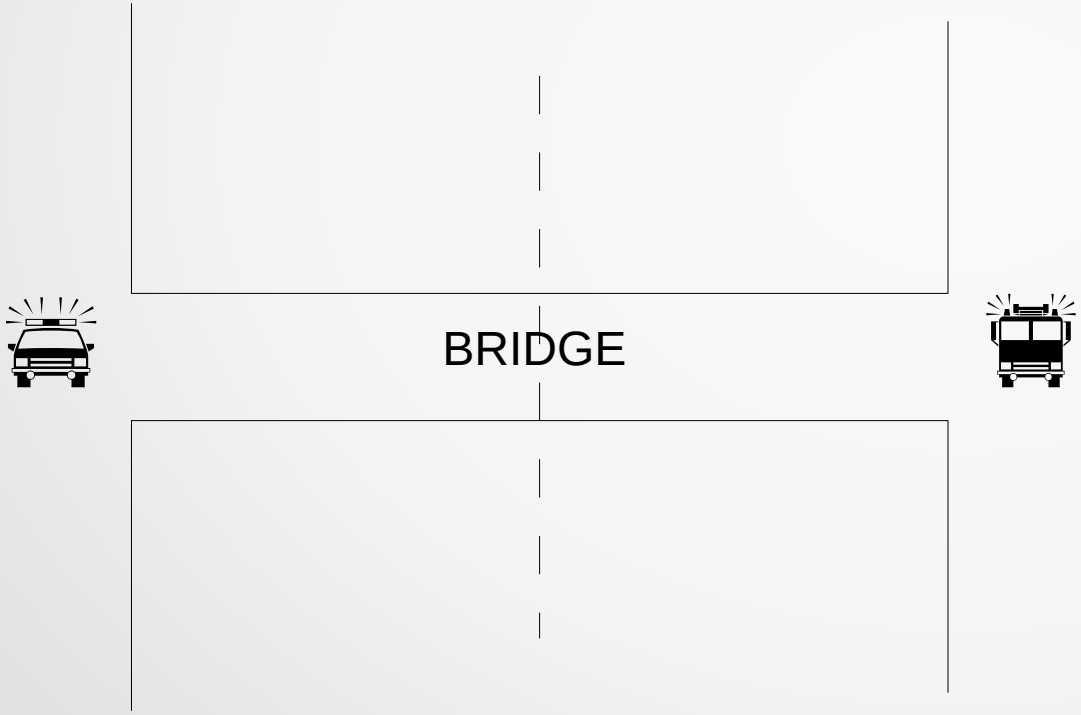
- Let's assume a system model, where:
  - There are *m* kinds of resources: Z(1), Z(2), … , Z(*m*).
  - There are E(*i*) (*i*=1...*m*) copies of each resource.
  - Resources are not interchangable.
    - In general case, resources may be interchangable.
- Processes in a system may:
  - Request a resource
  - Use an assigned resource
  - Release an assigned resource

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

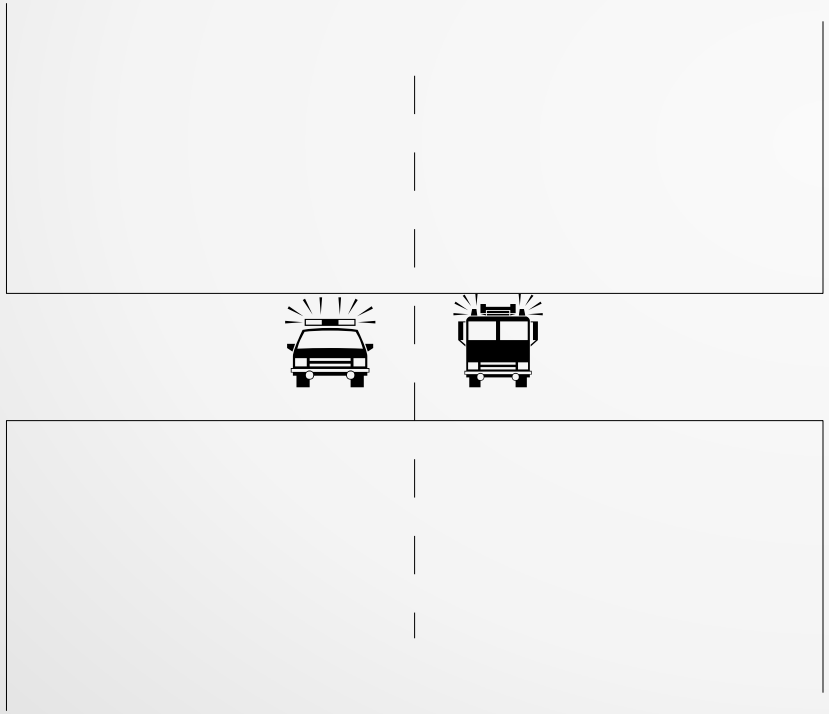
Deadlock

- **Deadlock** is a set of processes, which cannot continue their execution because other processes from this set occupy required resources.



Deadlock

- **Deadlock** is a set of processes, which cannot continue their execution because other processes from this set occupy required resources.



- Each car occupies its half of the bridge.
- None of them can pass due to the width of this bridge – they cannot simply pass by one another.
- One of them must give up and withdraw – which one?

Necessary conditions

- The following conditions must be fulfilled for a deadlock to happen:
  - **Mutual exclusion –** only one process may use the given resource at a time.
  - **Hold and wait** – a process that has at least one resource assigned waits for resources posessed by other processes.
  - **No preemption** – processes release their resources after they finished their job and there is no other way to take back the assigned resources.
  - **Cyclic waiting** – there exists a set of processes {P(1), …, P(n)} such that:
    - P(1) waits for resources assigned to P(2),
    - P(2) waits for resources assigned to P(3),
    - …
    - P(n) waiths for resources assigned to P(1).

Summary

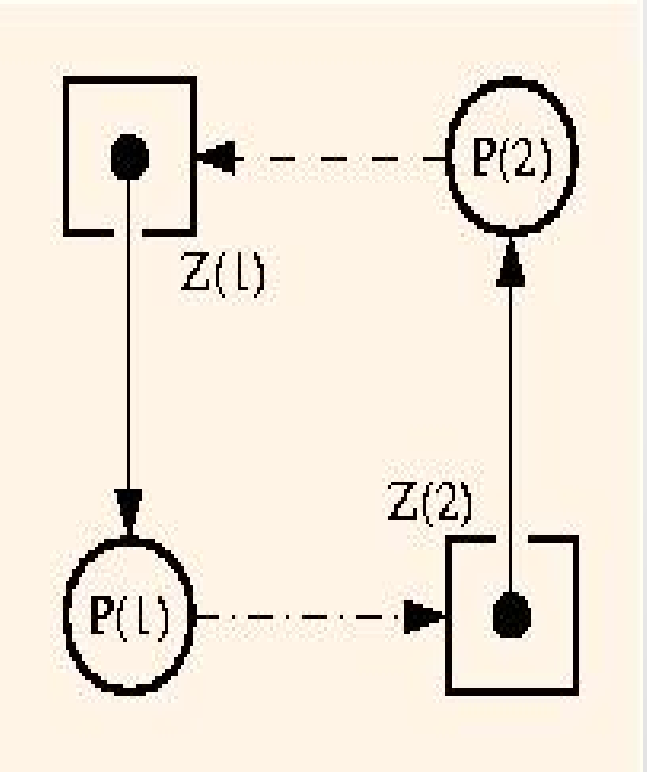
- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Resource allocation graph

- **Resource Allocation Graph** is a **directed** graph.
- It contains **two** kinds of nodes:
  - Nodes representing system resouces,
  - Nodes representing processes in a system.
- Edges represent one of:
  - resource-to-process assignment – an egde **from** a resource node **to** a process node.
  - Process-to-resource request – an egde **from** a process node **to** a resource node.

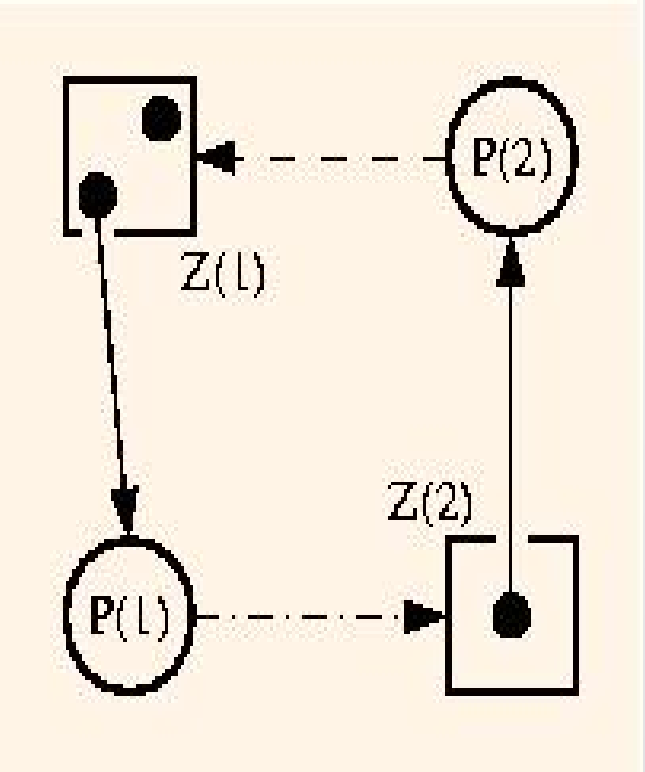
Resource allocation graph

- In the picture:
  - There ate 2 processes: P(1) and P(2)
  - There are two resources: Z(1) and Z(2)
  - There is **one** item of Z(1) and **one** item od Z(2).
  - Z(1) is assigned to P(1)
  - Z(2) is assigned to P(2).
- **If there is a cycle in a resource allocation graph, it means, thet there <u>may be</u> a deadlock.**



Resource allocation graph

- The deadlock may not exist, if the resources are in more than 1 piece – in the following picture there is no deadlock.



Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Deadlock prevention

- Deadlock prevention consists in such system design, that **at least one** of the necessary conditions does not exist:
  - Elimination of mutual exclusion is equal to solution of a problem. Unfortunately, using most of resources requires mutual exclusion, so this solution is usually not available.
  - No „hold and wait” may be organized for instance in such way, that the process must request for the whole set of needed resources at once. But it may result in poor resource usage and starvation of processes which need a lot of resources.

Deadlock prevention

- Preemption may be organized in such way, that a process requesting a resource which is not currently available, must return all the resources it posesses. Then it is put into a waiting queue. It will be released only if all the required resources are available and then this process will get them.
- Exclusion of cyclic waiting may be achieved by numbering of all the types of resources and introduction of a rule, that a process must request resources in an ascending order of resource numbers (i.e. It cannot apply for resource #4 after it was assigned resource #7).

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Deadlock avoidance

Avoidance is not prevention.

Safe system state

- Let's assume, that all the processes in the system know their maximal resource requirements and they report it to the system when they are started.
- The system tracks current resource usage for each process. It can fulfill its requests only to the limits implied by the reported maximal requirements.

Safe system state

- A system running a set of *n* processes is in a safe state, if there exists such sequence P(1), P(2), … , P(*n*) that requirements of process P(*i*) can be fulfilled by currently avalialbe resources and resources in posession of processes P(1) to P(*i*-1). This sequence is called a **safe sequence**.
  - We assume, that after a process is finished, all resources it posessed are freed and they can be reused by other processes.
  - So, P(1) can be executed using only available resources, P(2) can then use also resources occupied by P(1), and so on.
- If a system is in a safe state, there is no deadlock because there exists a safe order of process execution.

Safe system state

- An unsafe state does not always mean deadlock – a process doesn't have to request all the declared resources.
- The system tries to always stay in a safe state. It means, that whenever a process places a request for resources, it must be checked, if it does not make the system enter unsafe state.

The Banker's algorithm

- The algorithm consists in building a safe sequence of processes, based on their reported maximal requirements and current resouce usage.
- Assumption:
  - *n* processes, P(1), …, P(*n*).
  - *m* resources Z(1), …, Z(*m*).

The Banker's algorithm

- The algorithm uses the following arrays:
  - **Available[1...*m*]** – this array contains the number of currently available resource (Available[*i*] corresponds to resource Z(*i*)).
  - **Max[1...*n*, 1...*m*]** – maximal resource requirements reported by a process. For any *i*, *j*, Max[*i*, *j*] must not exceed Available[*j*].
  - **Assigned**[**1...*n*, 1...*m*]** – currently assigned resources for a process.
  - **Needs[1...*n*, 1...*m*]** – the number of resources, which a process may request from a system. In is assumed, that for any *i* and *j*:

$$\text{Max}[i, j] = \text{Assigned}[i, j] + \text{Needs}[i, j]$$

  - **End[1...*n*]** – End[*i*] == true only f process P(*i*) has finished its execution.
  - **Working[1...*m*]** – available resources in a system after some processes have finished their execution.

State safety detection

1. for *i*=1 to *n* do End[*i*] = false

   for *j*=1 to *m* do Working[*j*] = Available[*j*]
2. find *i* such that End[*i*]==false and Needs[*i*,*j*] <= Working[*j*] for all *j*.
3. If there is no such *i*, go to 6.
4. Assume, that P(*i*) is the next process in the safe sequence. After it is terminated, it returns its resources:

   Working[*j*] = Working[*j*] + Assigned[*i*,*j*] for all *j*.

   End[*i*] = true
5. Go to 2.
6. If for each *i* End[*i*] == true, the state is safe. Otherwise the state is not safe and may lead to deadlock.

Process request management

- Assume, that process P(*i*) has placed a resource allocation request represented as an array Request[1...*m*]. Request[*j*] is the number of resources of type Z(*j*) requested by P(*i*).
- If Request[*j*] > Needs[*i*,*j*] for any *j*, the demands exceeds the declared limit. Such request is then rejected.
- If Request[*j*] > Available[*j*] for any *j*, the demands exceeds the number of currently available resources. Such request is then rejected.

Process request management (cont.)

- Otherwise, the new state is created, according to the following rules:

  Available[*j*] = Available[*j*] – Request[*j*] for all *j*.

  Assigned[*i*,*j*] = Assigned[*i*,*j*] + Request[*j*] for all *j*.

  Needs[*j*] = Needs[*j*] – Request[*j*] for all *j*.

  - If such state is safe (according to the banker's algorithm), the request is accepted and resources are assigned.
  - If not, the request is rejected and the process must deal with it somehow – <u>although there are available resources!</u>

Example

- Let's assume a system with 3 processes P(1), P(2) and P(3) and 2 types of resources Z(1) and Z(2), with 3 items each. The corresponding arrays are shown in the array below.
- The state is safe, the processes may be executed in the following order: P(3), P(2), P(1).

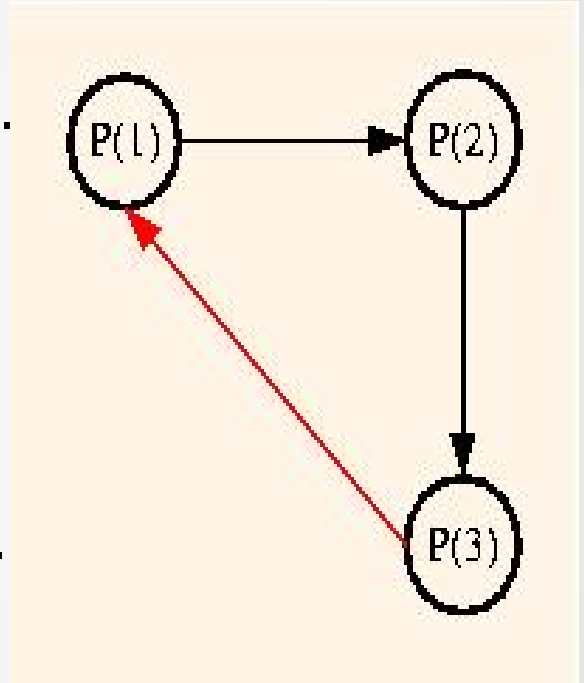
Process	Assigned		Max		Available	
	Z(1)	Z(2)	Z(1)	Z(2)	Z(1)	Z(2)
P(1)	0	0	2	1	1	1
P(2)	2	1	2	3		
P(3)	0	1	1	1		

Summary

- This lecture shows the problem of deadlocks:
  - Resources and system model.
  - Definition of deadlock.
  - Resource allocation graph
  - Deadlock management
    - Deadlock prevention
    - Deadlock avoidance
    - Deadlock detection and reconstruction

Deadlock detection

- We may allow deadlocks to happen in a system. But then, we must be able to detect it, remove it and resume correct system operation.
- In a system with single resources, we may use a „waiting graph”, which is a simplified form of resource allocation graph:
  - It is a directed graph.
  - Nodes correspond to processes in a system.
  - Edge between nodes P(*i*) and P(*j*) means, that process P(*i*) waits for resource in posesion of process P(*j*).
  - A cycle in such graph implies a deadlock.



Banker's algorithm for deadlock detection

- The banker's algorithm can be used for deadlock detection:
  - Select the processes, which have any resources assigned.
  - Use the banker's algorithm to check, if the system is in a safe state, if limited to only this subset of processes by finding the safe sequence of processes.
  - If there is a safe sequence of processes, there is no deadlock. If such order does not exist, there is a deadlock.

Resuming from deadlock

- After a deadlock is detected, it must be removed and the system must be put back into a safe state.
- Removal of a deadlock may consist in:
  - Killing all the processes in deadlock (to hard).
  - Kill processes one after another, until the deadlock is removed. It must be done in a proper order.
    - Time priority.
    - Execution time and estimated remaining process execution time.
    - Assigned resources or resources needed for process completion.
    - How many processes must be terminated.
    - Is it a batch or interactive process.
  - In any case, the system should minimise the cost of such operation (do not kill a process, which is to finish soon, etc.)

Thank You