Operating Systems

Lecture #7: Deadlock Resolution

Written by David Goodwin
based on the lecture series of Dr. Dayou Li
and the book *Understanding Operating Systems* 4^{th} *ed.*
by *I.M.Flynn and A.McIver McHoes* (2006)

Department of Computer Science and Technology,
University of Bedfordshire.

Operating Systems, 2013

11^{th} March 2013

Outline

David Goodwin
University of Bedfordshire

David Goodwin
University of Bedfordshire

Introduction

Learning Objectives

David Goodwin
University of Bedfordshire

- Several causes of system deadlock.
- The difference between preventing and avoiding deadlocks.
- How to detect and recover from deadlocks.
- The concept of process starvation and how to detect and recover from it.
- The concept of a race, and how to prevent it.
- The difference between deadlock, starvation, and race.

Deadlock

DEADLOCK

- A situation (problem) where resources needed by some processes to finish execution are held by other processes which, in turn, are waiting for other resources to become available
- A lack of **process syncronisation** can result in two extreme conditions:
  1. deadlock
  2. starvation
- Deadlock is a system wide tangle of resource requests
  - begins when two or more jobs are put on hold
  - each waits for a vital resource to become available
  - problem builds when resources needed are those held by other jobs waiting for other unavailable resources
  - jobs come to a standstill
  - deadlock is complete when the rest of the system comes to a standstill as well
- deadlock infrequent in batch systems, and more prevalent in interactive systems

The 7 Cases

The seven cases of deadlock

1. Deadlocks on file requests
2. Deadlocks in databases
3. Deadlocks in dedicated device allocation
4. Deadlocks in multiple device allocation
5. Deadlocks in spooling
6. Deadlocks in a network
7. Deadlocks in disk sharing (livelock)

DEADLOCK - case I

- Case 1: Deadlocks on file requests
  - If processes are allowed to request and hold files for the duration of their execution, deadlocks can occur
- Example:
  - P1 requires raw data from R1.dat file and sends results to R2.dat file
  - P2 requires raw data from R2.dat file and sends results to R1.dat file
  - P1 and P2 have to work simultaneously

DEADLOCK - case II

- Case 2: Deadlocks in databases
  - Deadlock can also occur when two processes try access and lock the same records in a database (**locking** is a technique used to guarantee the integrity of the data)
- Example:
  - Java program 1 (JP1) needs to update pages and authors for one record in Pages and Authors data-tables within DB
  - Java program 2 (JP2) also needs to update pages and authors for another record in both data-table of the DB
  - JP1 holds and locks Pages data-table
  - JP2 holds and lock Authors data-table
  - JP1 requests Authors table to finish
  - JP2 requests Pages table to finish
  - ...
- If locks aren't used, the database may only include some of the data, and the contents would depend on the order in which process finishes its execution. This is known as a **race**.

DEADLOCK - case III

David Goodwin
University of Bedfordshire

- Case 3: Deadlocks in dedicated device allocation
  - The use of a group of dedicated devices can also deadlock the system
- Example:
  - P1 requests disk drive 1 and gets it
  - P2 requests disk drive 2 and gets it
  - P1 also requests disk drive 2 but is blocked by P2
  - P2 also requests disk drive 1 but is blocked by P1

DEADLOCK - case IV

David Goodwin
University of Bedfordshire

- Case 4: Deadlocks in multiple device allocation
  - Deadlocks aren't restricted to processes contending for the same type of device, they can happen when several processes request and hold on to dedicated devices, while other processes act in a similar manner.
- Example:
  - P1 requests CD drive and gets it
  - P2 requests printer and gets it
  - P3 requests potter and gets it
  - P1 also requests printer but is blocked by P2
  - P2 also requests plotter but is blocked by P3
  - P3 also requests CD drive but is blocked by P1

DEADLOCK - case V

David Goodwin
University of Bedfordshire

- Case 5: Deadlocks in spooling
  - Spooling – a high-speed disk serves as a temporary storage in printer server for files sent by multiple users, so a “batch” of files will be given to the printer
  - Deadlock occurs when the disk is full with interim outputs from the users but non final outputs – the printer cannot work as no final output is available

DEADLOCK - case VI

- Case 6: Deadlocks in network
  - network can be deadlocked when it is congested or it has filled a large percentage of its I/O buffer space
- Example:
  - C1 recieves messages from nodes C2, C6 and C7
  - C1 sends message to C2
  - C2 recieves messages from C1, C3, and C4
  - C2 sends messages to C1 and C3
  - messages recieved by C1 from C6 and C7 destined for C2, are buffered in output queue
  - messages recieved by C2 from C3 and C4 destined for C1, are buffered in output queue
  - at high traffic, buffer space is filled
  - C1 cant accept more
  - C2 cant accept more
  - communication between C1 and C2 is deadlocked

DEADLOCK - case VII

- Case 7: Deadlocks in shared disk
  - **Livelock** occurs when P1 and P2 try to access a shared disk through the same I/O channel but not gaining any control over the I/O channel
- Example:
  - P1 issues a READ command on a disk at cylinder 20 of the disk pack
  - While the control unit is moving the arm to cylinder 20, P1 is put on hold and the IO channel is free to process the next IO request
  - P2 gains control of the IO channel and issues WRITE command on the disk at cylinder 310 of the disk pack. P2 will be put on hold until the arm moves to cylinder 310
  - The channel is free and so captured by P1, which reconfirms the command READ cylinder 20
  - The arm is in constant motion, moving back and forth between cylinder 20 and 310, responds to two competing commands but satisfyies neither.

Conditions

The conditions for deadlock

1. **Mutual exclusion**
   - the act of allowing only one process to have access to a dedicated resource (Cases 1, 2, 7)
2. **Resource holding**
   - a number of processes all hold resources requested by others but none give up (Cases 1, 2, 3, 4, 6, 7)
3. **No preemption**
   - the lack of temporary reallocation of resources (Case 5)
4. **Circular wait**
   - each process involved is waiting for another to release the resource, so that at least one process will be able to be completed (Case 2)

The conditions for deadlock

- All four conditions are required for the deadlock to occur
- As long as all four conditions are present, deadlock will continue
- If one condition can be removed the deadlock will be resolved
- If the four conditions can be prevented from occuring at the same time, deadlock can be prevented (not easy to implement)

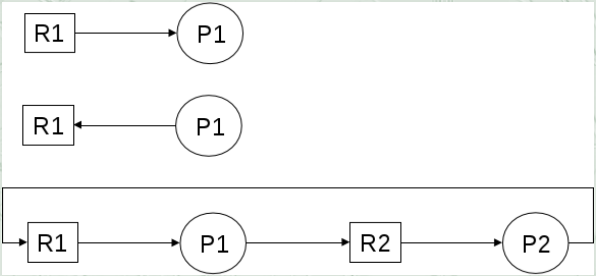
Modelling

DEADLOCK MODELLING

- Graph
  - Graph is a maths term and also a data structure which contains a number of nodes linked with edges
  - Directed graph has all its edges associated with arrows representing information flow
  - Directed graph used for deadlock modelling
    - Circle: a type of node representing a process
    - Rectangle: a type of node representing a resource
    - Edge with arrow from a circle to a rectangle: the process requesting the resource
    - Edge with arrow from a rectangle to a circle: the process holding the resource
    - No connection: resource is released
    - A cycle in a directed graph: deadlock

DEADLOCK MODELLING

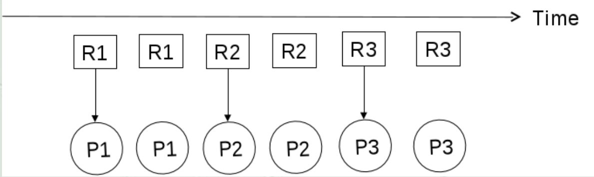
David Goodwin
University of Bedfordshire



DEADLOCK MODELLING

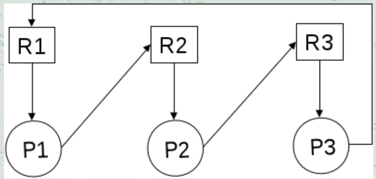
David Goodwin
University of Bedfordshire

- Example 1
  - P1, P2 and P3
  - R1, R2 and R3
  - Scenario 1 (not locked):
    - P1 request R1 and holds R1
    - P1 releases R1
    - P2 request R2 and holds R2
    - P2 releases R2
    - P3 request R3 and holds R3
    - P3 releases R3



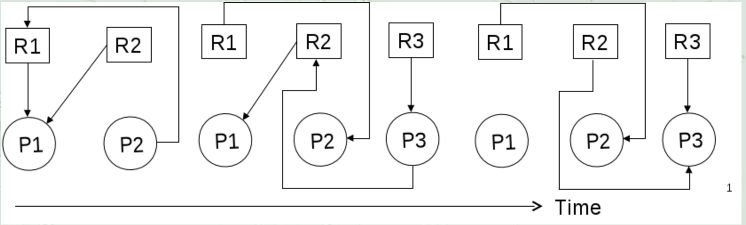
DEADLOCK MODELLING

-
    - Scenario 2 (deadlock):
        - P1 request R1 and holds R1
        - P2 request R2 and holds R2
        - P3 request R3 and holds R3
        - P1 requests R2
        - P2 requests R3
        - P3 requests R1



DEADLOCK MODELLING

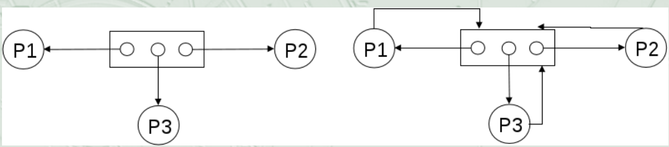
- Scenario 3:
    - P1 request R1 and holds R1
    - P1 request R2 and holds R2
    - P2 request R1
    - P3 request R3 and holds R3
    - P1 releases R1 which is then allocated to P2
    - P3 requests R2
    - P1 releases R2 which is then allocated to P3



DEADLOCK MODELLING

- Expansion of directed graph
  - A group of the same type of resources can be represented as:

    (A group of three resources of the same type)
  - Deadlock will occur if P1, P2 and P3 each hold one resources in the group but request more



Handling

Strategies for handling deadlock

- In general there are three strategies to deal with deadlock:
  - **Prevention**
    - Prevent one of the four conditions of deadlock
  - **Avoidance**
    - Avoid deadlock if it becomes probable
  - **Detection**
    - Detect deadlock when it occurs and recover from it gracefully

DEADLOCK HANDLING - Prevention

- Prevention
  - An O/S must eliminate any one of four conditions
    - Mutual exclusion
    - Resources holding
    - No pre-emption
    - Circular wait

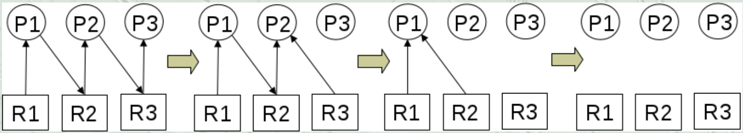
DEADLOCK HANDLING - Detection

David Goodwin
University of Bedfordshire

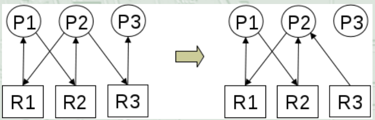
- Detection
  - Detection algorithm detects the sign of deadlock by reducing a directed graph
    - Removing edges associated with processes that are not requesting any new resource
    - Removing edges associated with processes that only request resources
    - If any edge left, then deadlock could take place

DEADLOCK HANDLING - Detection

- Example 2: edges can be removed



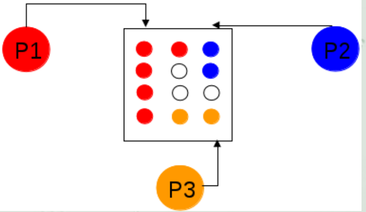
- Example 3: edges cannot be removed



DEADLOCK HANDLING - Avoidance

- Avoidance
  - Safe state and unsafe state
    - Safe sequence $[p_1, p_2, \ldots p_i, \ldots p_n]$: a sequence of processes where p_i's request for new resources can be satisfied by the currently available ones plus those held by p_j $(j < i)$
    - A system's safe state: if there is a safe sequence in the system
  - Example: 12 types and three processes

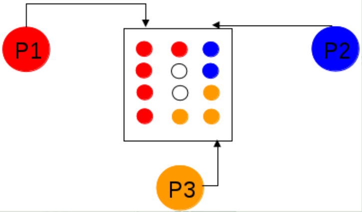
	maximal needed	currently needed
P1	10	5
P2	4	2
P3	9	2



DEADLOCK HANDLING - Avoidance

- System is safe if the sequence is $[p_2, p_1, p_3]$: p_2 requests 2 more and then releases 4, p_1 obtains all 5 and then releases all, p_3 requests 7
- What about $[p_2, p_3, p_1]$?
- System is unsafe if p_3 is allocated one more: p_2 asks 2 and gets them. p_2 releases 4; p_1 requests 5 but only 4 are available.

	maximal needed	currently needed
P1	10	5
P2	4	2
P3	9	3



DEADLOCK HANDLING - Banker's algorithm

David Goodwin
University of Bedfordshire

- Banker's algorithm
  - When a process enters a system, it must declare the max resources it will need, the system decides whether to allocate the resources depends on whether the allocation of resources will lead the system to a safe state
  - Data structures needed in Banker's algorithm:
    - Array ava[m] – available resources of type m: ava[j]=k means k resources of type j are available
    - Matrix max[n][m] – total resource of type j needed by process Pi: max[i][j]=k means Pi request the most k resources of type j
    - Matrix all[n][m] – held resources of type j by process Pi: all[i][j]=h means pi is currently allocated h resources of type j
    - Matrix need[n][m] – Pi will need i resources of type j: need[i][j]=i means Pi may need i more resources of type j to complete

DEADLOCK HANDLING - Safety algorithm

David Goodwin
University of Bedfordshire

- Safety algorithm
  1. Define int work[m] and bool fini[n] and work[i]=ava[i] for all i, and fini[i]=false for all i
  2. Find an index i such that
     fini[i]==false and need[i]¡= work[i]
     If no such an i, go to 4
  3. work[i]=work[i]+all[i]
     fini[i]=true
     Goto 2
  4. If(fini[i]==true) for all i, then the system is in a safe state

DEADLOCK HANDLING - Resource-request algorithm

David Goodwin
University of Bedfordshire

- Resource-request algorithm
  1. Loop (j = 1, j++, j¡=n)
  2. Define int req[i] – resources of type j requested by Pi
  3. If req[i]¡=need[i][j] go to 2, otherwise, raise an error condition
  4. If req[i]¡=ava[j], go to 3, otherwise, pi must wait
  5. (after Pi finishes) update ava, all and need
     ava[i]=ava[i]+req[i]
     all[i]=0
     need[i][j]=need[i][j]-req[i]

DEADLOCK RECOVERY

- Once a deadlock has been detected it must be recovered
- There are several algorithms that have one feature in common: they all require a **victim**, an expendable job, which when removed from the deadlock, will free the system.
- Deadlock Recover
  - Terminate all processes
  - Terminate all processes which are involved in deadlock
  - Terminate all processes which are involved in deadlock one at a time
  - Interrupt a process that keeps a record or snapshot
- Factors considered when selecting a victim
  - Priority
  - CPU time
  - Number of other jobs affected

SUMMARY

key terms

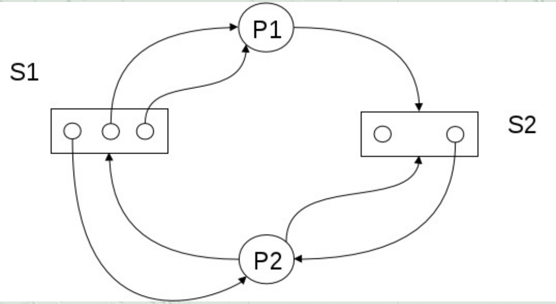
avoidance
circular wait
deadlock
detection
directed graphs
livelock
locking
mutual exclusion
no preemption

key terms

prevention
process synchronisation
race
recovery
resource holding
safe state
spooling
starvation
unsafe state
victim

Exercise 1

- Exercise 1:
  - Is there any deadlock in the following graph?



Exercise 2

- Exercise 2:
  - Is there any deadlock in the following graph?