Lecture Overview

- Deadlocks
  - More thorough introduction to deadlocks
  - Deadlock modeling
  - Dealing with deadlocks
    - The ostrich approach
    - Detection and recovery
    - Avoidance
    - Prevention

Sharing Everywhere

- The OS is the maintainer of a numerous different types of resources
- Numerous processes and threads can exist within the OS that all want access to the same resources
- The OS is responsible for enabling sharing of resources
  - We saw some process coordination primitives that enabled proper sharing among process
  - The OS must use these primitives and other techniques to ensure that access to its resources remain consistent
  - The thread of deadlock in the OS is great

Sharable Resources

- Examples of sharable computer resources
  - Printers
  - Tape drives
  - Tables
- Preemptable resources
  - Can be taken away from a process with no ill effects
- Non-preemptable resources
  - Will cause the process to fail if taken away
  - *We are concerned with this type of resource*

Sharable Resources

- The OS must provide must provide exclusive access to non-preemptable sharable resources
- Sequence of events required to use a resource
  - *Request* the resource
  - *Use* the resource
  - *Release* the resource
- When a process wants to use a resource that is already being used by another process
  - Requesting process may be blocked
  - May fail with error code

Deadlock

- Deadlocks occur when …
  – Processes are granted exclusive access to resources
- Formal definition

  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
- Usually the event is the release of a held resource
- When deadlocked, none of the processes can
  – Run
  – Release resources
  – Be awakened

Four Condition for Deadlock

- Mutual exclusion condition
  – Each resource can only be assigned to at most one process at a time
- Hold and wait condition
  – Processes holding resources can request additional resources
- No preemption condition
  – Previously granted resources cannot forcibly taken away
- Circular wait condition
  – Must be a circular chain of 2 or more processes
  – Each is waiting for resource held by next member of the chain

Strategies for Dealing with Deadlock

- The OS should be written carefully so that it will never deadlock, but processes can still deadlock
- Approaches for OS to deal with process deadlock
  - Ignore the problem altogether
  - Detection and recovery
  - Dynamic avoidance
    - Careful resource allocation
  - Prevention
    - Negating one of the four necessary conditions

Strategies for Dealing with Deadlock

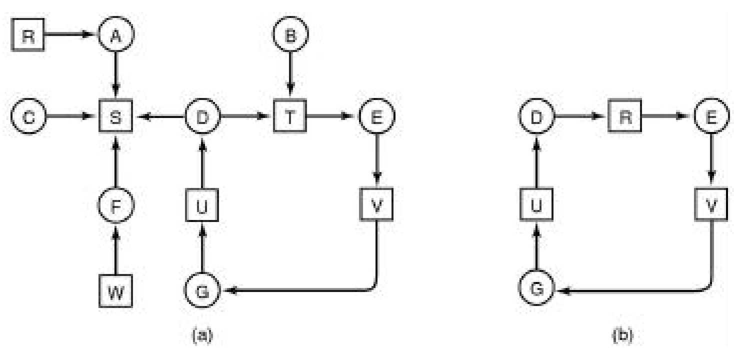
- Ignore the problem altogether
  - Pretend there is no problem
  - Reasonable if
    - Deadlocks occur very rarely
    - Cost of prevention is high
  - It is a trade off between
    - Convenience
    - Correctness
  - UNIX and Windows takes this approach as do most OSs
    - If no OSs deal with deadlock, why study it?
      - We at least need to understand the issues

Strategies for Dealing with Deadlock

- Deadlock detection and recovery
  - The system lets deadlocks occur
  - The system tries to detect when a deadlock occurs
  - When a deadlock is detected, some action is taken to try to recover from it
  - One technique for detecting deadlocks is to build a resource graph
    - A circle represents a process
    - A square represents a resource
    - A directed arc from a resource to a process denotes ownership of a resource
    - A directed arc from a process to a resource denotes a request for a resource

Strategies for Dealing with Deadlock

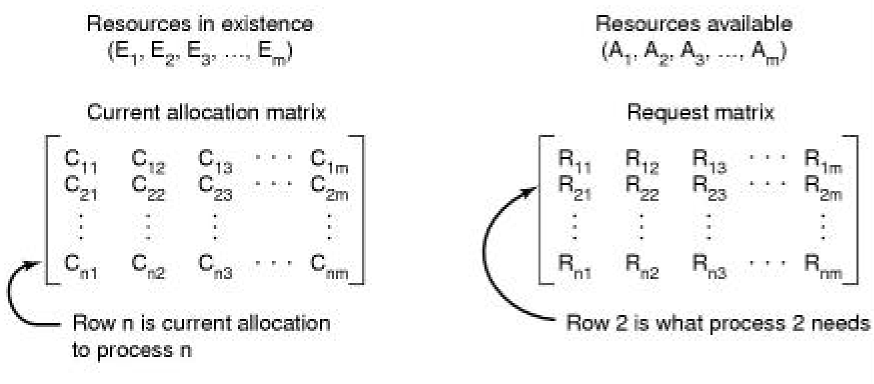
- Deadlock detection and recovery



  - Note the resource ownership and requests
  - A cycle can be found within the graph, denoting deadlock
  - This is for one resource of each type, but can be extended

Strategies for Dealing with Deadlock

- Deadlock detection and recovery
  - A different approach to deal with multiple instances of multiple resource types



Strategies for Dealing with Deadlock

- Deadlock detection and recovery
  - Using the previous tables, deadlock detection is based on comparing vectors
    - The relation *X* £ *Y* on two vectors *X* and *Y* means that each element of *X* is less than or equal to the corresponding element in *Y*
    - Each process is initially defined as unmarked
    - The detection algorithm
      - Looks for an unmarked process, P_i, for which the *i*-th row of the *R* table is £ the *A* vector
      - If such a process is found, add the *i*-th row of the *C* table to the *A* vector, mark the process, and go back to first step
      - If no such process exists, the algorithm terminates
      - Any unmarked processes are known to be deadlocked

Strategies for Dealing with Deadlock

- Deadlock detection and recovery
  - Example

Tape drives, Plotters, Scanners, CD Roms

$$E = (4 \quad 2 \quad 3 \quad 1)$$

Tape drives, Plotters, Scanners, CD Roms

$$A = (2 \quad 1 \quad 0 \quad 0)$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Strategies for Dealing with Deadlock

- Deadlock detection and recovery
  - Recovering from a detected deadlock
    - Recovery through preemption
      - Take a resource from some other process
      - Depends on nature of the resource
    - Recovery through rollback
      - Checkpoint a process periodically
      - Use this saved state
      - Restart the process if it is found deadlocked
    - Recovery through killing processes
      - Crudest but simplest way to break a deadlock
      - Kill one of the processes in the deadlock cycle
      - Choose a process that can be rerun from the beginning

Strategies for Dealing with Deadlock

- Deadlock avoidance
  - Make it impossible for deadlocks to occur at all
  - Processes must announce maximum resource requirements in advance
  - Use the tables we already define in deadlock detection
    - The current state of a system consists of the values of *E*, *A*, *C*, and *R*
    - A state is said to be *safe* if it is not deadlocked and there is some scheduling order in which every process can run to completion even if every process requests their maximum amount of resources
    - The system never allows itself to enter an *unsafe* state

Strategies for Dealing with Deadlock

- Deadlock avoidance
  - Example of a safe state with multiple instances of a single resource type
    - Use same steps as deadlock detection
    - Do not allow a request that would lead to unsafe state
      - Always check next state using detection algorithm before granting resources

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Strategies for Dealing with Deadlock

- Deadlock avoidance
  - Example of an unsafe state with multiple instances of a single resource type; process A requests another resource

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Strategies for Dealing with Deadlock

- Deadlock avoidance
  - The Banker's Algorithm for multiple resource types
    - Before granting request, perform same steps as deadlock detection to check whether next state is safe
      - Look for row R whose unmet resource needs are all smaller than or equal to A
      - If no row exists, the state is unsafe
      - If a row exists, assume the process of row R requests and then finishes, releasing all of its resources; mark the process as finished and add its resources to vector A
      - Repeat until all processes are marked as finished (state is safe) or the state is determined to be unsafe

Strategies for Dealing with Deadlock

- Deadlock avoidance
  - The Banker's Algorithm for multiple resource types

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Strategies for Dealing with Deadlock

- Deadlock prevention
  - Try to eliminate one of the four conditions of deadlock
    - Mutual exclusion
      - Not really possible in many cases since some resources require mutual exclusion (e.g., a printer)
    - Hold and wait condition
      - Only allocate all resources at once
      - Only allocate resource when process has none
      - This results in poor resource utilitization
      - Starvation is possible
    - No preemption
      - Preempt resources from processes that block (e.g., the CPU)
      - This is not possible for all types of resources
    - Circular wait
      - Impose total ordering on all resources

Deadlock Conclusions

- The potential for process deadlock is great in OS
- There are various approaches and mechanisms for dealing with deadlock
- Most OSs only guarantee mutually exclusive access to appropriate resources, but do not try to prevent processes from deadlocking