CSC 256/456: Operating Systems

Deadlock

John Criswell
University of Rochester

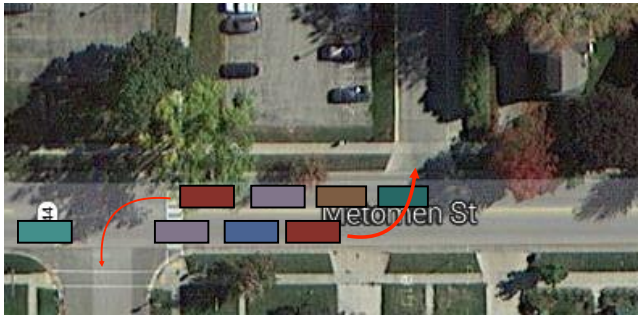
Outline

- Basics of deadlock
- Three approaches for handling deadlock

The Deadlock Problem

- Definition:
  - A set of blocked processes each holding some resources and waiting to acquire a resource held by another process in the set
  - None of the processes can proceed or back-off (release resources it owns)
- Examples:
  - Dining philosopher problem
  - System has 2 memory pages (unit of memory allocation); P1 and P2 each hold one page and each needs another one
  - Semaphores A and B, initialized to 1
    - P1 P2
    - wait (A); wait(B)
    - wait (B); wait(A)

Real Life Example from Metomen Street



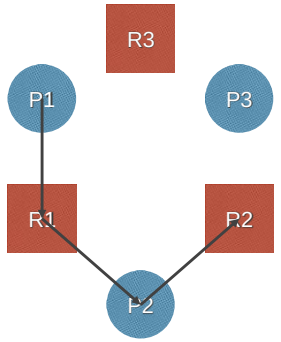
Resources Can Be Anything!

- Devices!
- Files!
- Networks!
- Mutexs!
- Sections of Road!
- Time itself!

5

Resource Allocation Graph

- Nodes
  - Processes
  - Resources (or resource classes)
- Request Edge
  - P -> R: Process Wants Resource
- Assignment Edge
  - R -> P: Resource held by process



6

The Four Conditions of Deadlock

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

7

Mutual Exclusion

- Only one process at a time can use a resource

8

Hold and Wait

- Process can hold one resource while waiting to acquire additional resources held by other processes

9

No Preemption

- Resource can be released only voluntarily by the process holding it, after that process has completed its task

10

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.

11

Methods for Handling Deadlocks

- Ignore the problem!
- Ensure that the system will never enter a deadlock state
  - Deadlock Prevention
  - Deadlock Avoidance
- Detect and recover from deadlock state

12

Of Ostriches and Resources

13

The “Ostrich” Algorithm

- Pretend there is no problem
  - unfortunately they can occur
- Reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- Typical OS take this approach *for processes and devices*
- Trade-off between *convenience* and *correctness*

14

Deadlock Prevention

15

Deadlock Prevention

- Establish conventions that prevent deadlock
- Prevents one of the four conditions from happening

16

Attack the Mutual Exclusion Condition!

- Don't permit mutual exclusion
  - Read-only files
  - Spooled devices (e.g., printer)
- Can't be done in most cases

17

Attack the Hold and Wait Condition!

- Require processes to request all resources before starting
- Problems
  - May not know required resources at start of run
  - Ties up resources other processes could be using
- Variation:
  - before a process requests a new resource, it must give up all resources and then request all resources needed

18

Attack the No Preemption Condition!

- Preemption
  - when a process is holding some resources and waiting for others, its resources may be preempted to be used by others
- Problem
  - Many resources may not allow preemption; i.e., preemption will cause process to fail

19

Attack the Circular Wait Condition!

- Impose a total order of all resource types
- Processes *must* request resources in the same order
- Often used for mutexs

20

Deadlock Avoidance

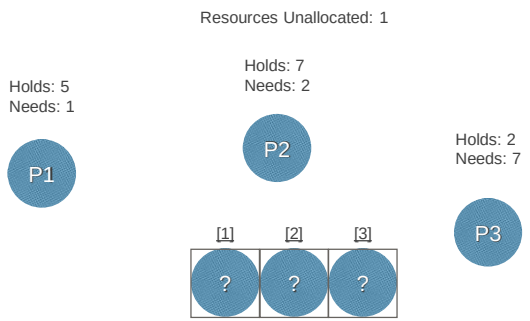
Prevention vs. Avoidance

- Prevention prevents conditions under which deadlock can occur
  - Enforces conventions that break deadlock conditions
- Avoidance refuses resource allocation if deadlock would be possible in the future
  - Examines state of allocation to check for future deadlock

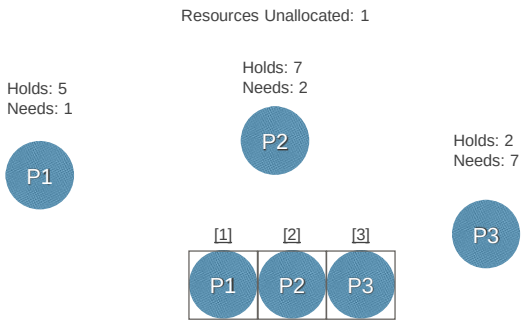
Deadlock Avoidance

- Processes declare max amount of resources ever needed
- How to determine if allocation state is deadlock free:
  - Line up processes in some order such that for all P_i:
    - Can allocate all future resources from either:
      - Free resources
      - Resources held by all P_j such that $j < i$
  - State of system is called a *safe state*

Deadlock Avoidance Example 1

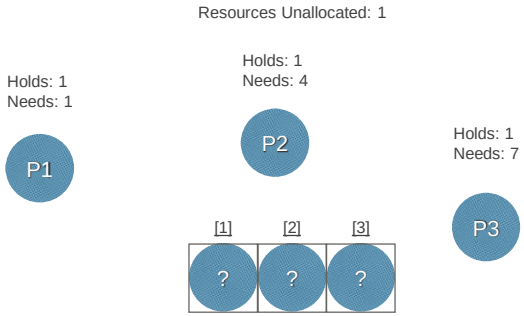


Deadlock Avoidance Example 1



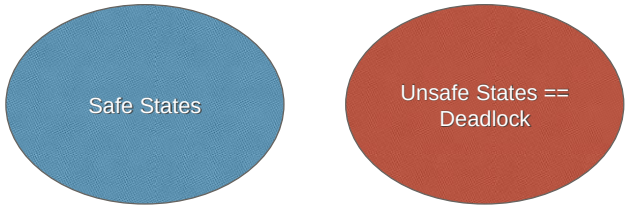
25

Deadlock Avoidance Example 2



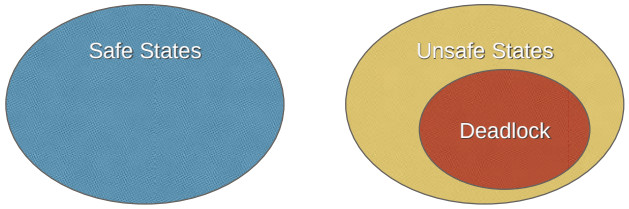
26

One Resource Per Resource Class



29

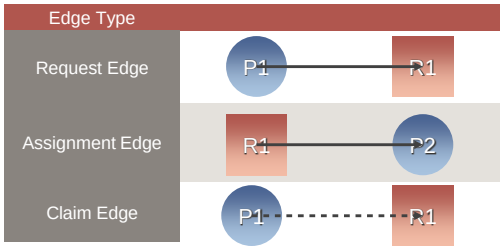
Multiples Resources Per Resource Class



30

Deadlock Prevention with Resource Allocation Graphs

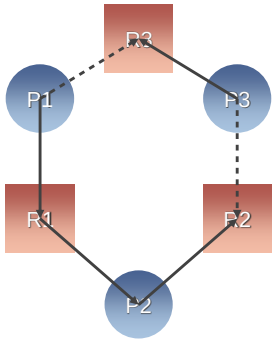
- Works with single resource per resource class
- Enhance allocation graph with three edges



32

Deadlock Prevention with Resource Allocation Graphs

- On allocation:
  - Change claim edge to assignment edge
  - Check for cycle
  - Cycle implies deadlock



33

Banker's Algorithm

- Each process must a priori claim the maximum set of resources that might be needed in its execution
- Safety check
  - repeat
    - pick any process that can finish with existing available resources; finish it and release all its resources
    - until no such process exists
  - all finished → safe; otherwise → unsafe.
- When a resource request is made, the process must wait if:
  - enough available resource is not available for this request
  - granting the request would result in an unsafe system state

35

Deadlock Detection

- Deadlock detection is very similar to the safety check in the Banker's algorithm
  - Replace maximum needs with current requests

39

Recovery from Deadlock

- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process state periodically
  - rollback a process to its checkpoint state if it is found deadlocked
- Recovery through killing processes
  - kill one or more of the processes in the deadlock cycle
  - the other processes get its resources
  - In which order should we choose process to kill?
- Will recovery lead to starvation?

Disclaimer