CSE 451: Operating Systems
Spring 2006

Module 8
Deadlock

John Zahorjan
zahorjan@cs.washington.edu
Allen Center 534

(Is Google the greatest, or what?)

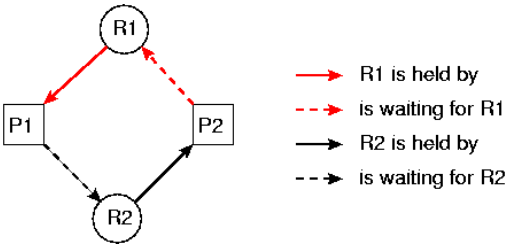
Definition

- A thread is deadlocked when it's waiting for an event that can never occur
  - I'm waiting for you to clear the intersection, so I can proceed
    - but you can't move until he moves, and he can't move until she moves, and she can't move until I move
  - thread A is in critical section 1, waiting for access to critical section 2; thread B is in critical section 2, waiting for access to critical section 1
  - I'm trying to book a vacation package to Tahiti – air transportation, ground transportation, hotel, side-trips. It's all-or-nothing – one high-level transaction – with the four databases locked in that order. You're trying to do the same thing in the opposite order.

Requirements

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

Resource graph

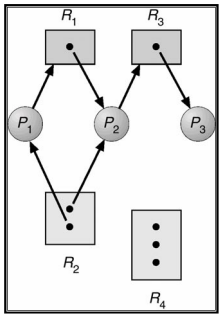


- A deadlock exists if there is an *irreducible cycle* in the resource graph (such as the one above)

Graph reduction

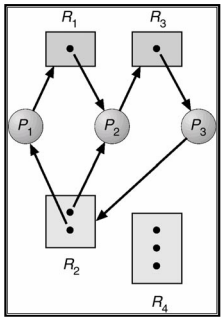
- A graph can be *reduced* by a thread if all of that thread's requests can be granted
  - in this case, the thread eventually will terminate – all resources are freed – all arcs (allocations) to it in the graph are deleted
- Miscellaneous theorems (Holt, Havender):
  - There are no deadlocked threads iff the graph is completely reducible
  - The order of reductions is irrelevant
- (Detail: resources with multiple units)

Resource allocation graph with no cycle

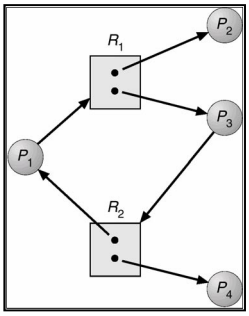


What would cause a deadlock?

Resource allocation graph with a deadlock



Resource allocation graph with a cycle but no deadlock



Approaches to Deadlock

- Break one of the four required conditions
  - Mutual Exclusion?
  - Hold and Wait?
  - No Preemption?
  - Circular Wait?
- Broadly classified as:
  - prevention, or
  - avoidance, or
  - detection (and recovery)

Prevention

- Hold and Wait
  - each thread obtains all resources at the beginning; blocks until all are available
    - drawback?
- Circular Wait
  - resources are numbered; each thread obtains them in sequence (which means acquiring some before they are actually needed)
    - why does this work?
    - pros and cons?
- Mutual Exclusion
  No Preemption
  - Application limited

Avoidance

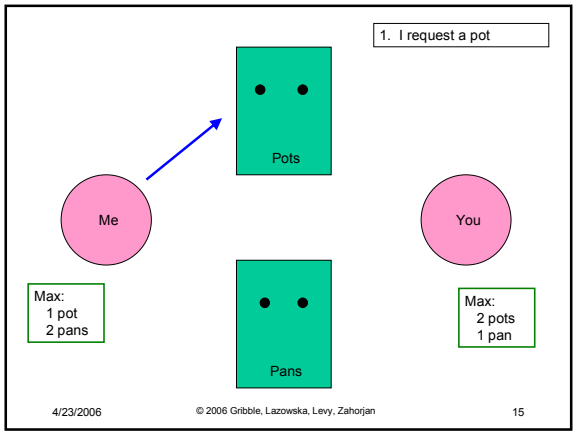
- Circular Wait
  - each thread states its maximum claim for every resource type
  - system runs the Banker's algorithm at each allocation request
    - Banker ⇒ incredibly conservative
    - if I were to allocate you that resource, and then everyone were to request their maximum claim for every resource, could I find a way to allocate remaining resources so that everyone finished?
  - More on this in a moment…

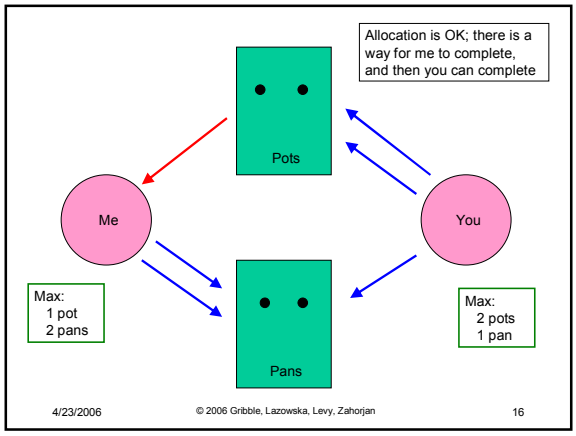
Detection and Recover

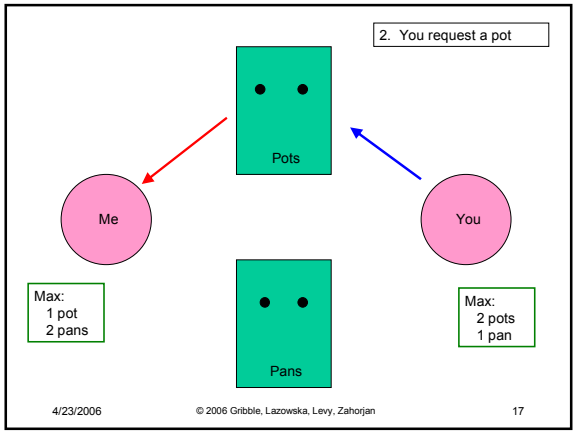
- every once in a while, check to see if there's a deadlock
  - how?
- if so, eliminate it
  - how?

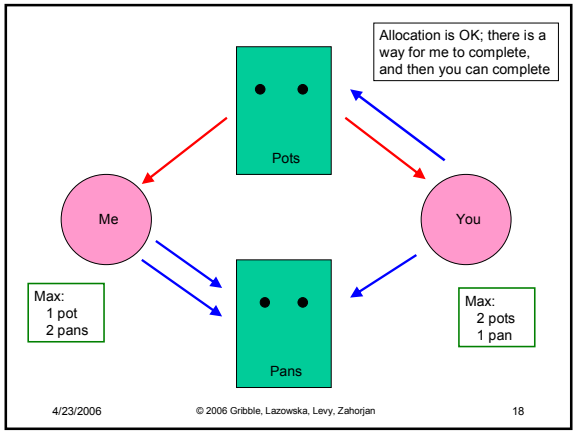
Avoidance: Banker's Algorithm Example

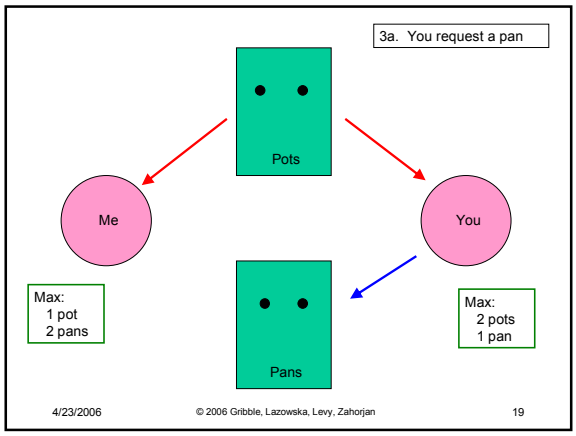
- When a request is made
  - pretend you granted it
  - pretend all other legal requests were made
  - can the graph be reduced?
    - if so, allocate the requested resource
    - if not, block the thread

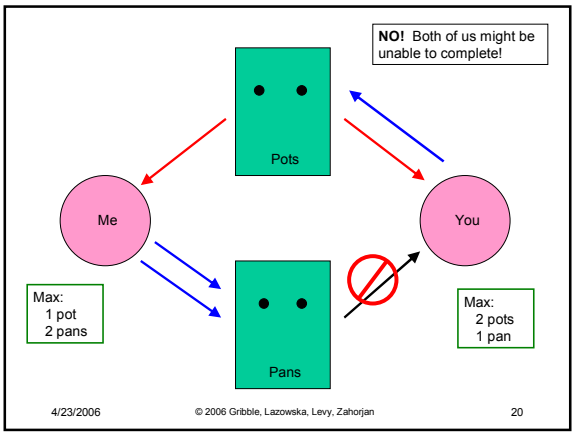


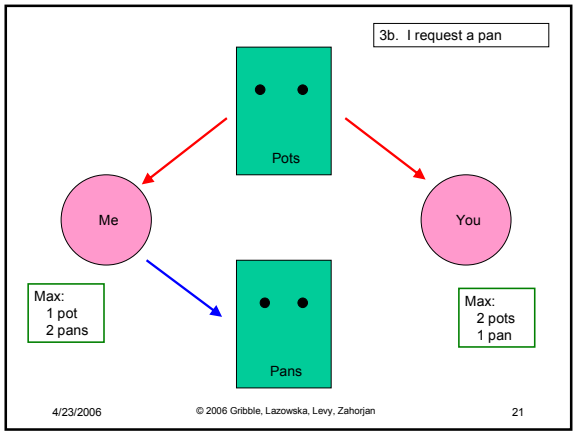


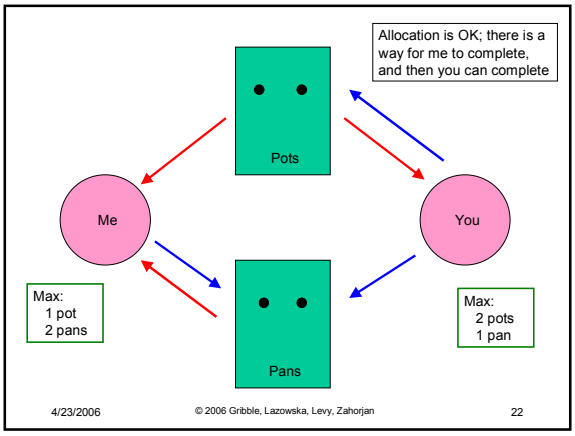












Summary

- Deadlock is bad!
- We can deal with it either statically (prevention) or dynamically (avoidance and detection)
- In practice, ordering locks is probably the technique you'll encounter most often

4/23/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 23