

CS 414 : Operating Systems

UNIVERSITY OF VIRGINIA
Department of Computer Science

Spring 2008

Topic 9: Deadlock

- Readings for this topic: Chapter 7
- Deadlock is one area where there is a lot of theory, but most of them are ignored in practice.
Why?
 - Simple approaches vs complex glorious approaches
- Deadlock example: semaphores.

- Define deadlock: a situation where each of a collection of processes is waiting for something from other processes in the collection. Since all are waiting, none can provide any of the things being waited for.

- How do you know there's a deadlock?

- Facts about deadlock:
 - The semaphore example is a relatively simple-minded case. Things may be much more complicated.
 - A deadlock can happen with any kind of non-preemptible resources.
 - In general, don't know the resource needs in advance. If only we could predict the future....
 - Deadlock can occur over separate resources, as in semaphore example, or over pieces of a single resource, as in memory, or even over totally separate classes of resources (tape drives and memory).
 - Whenever there is waiting, there's potential for deadlock. Hard for OS to control.

- Four conditions for deadlock:
 - Mutual exclusion: resources cannot be shared.
 - No preemption. once given, a resource cannot be taken away.
 - Hold and wait: processes don't ask for resources all at once (i.e., multiple independent requests).
 - Circular wait: circularity in the graph of who has what and who wants what.

- Solutions to the deadlock problem fall into three general categories:
 - Detection and resolution: determine when the system is deadlocked and then take drastic action. Requires termination of one or more processes in order to release their resources. Usually not very practical.
 - Prevention: organize the system so that it is impossible for deadlock ever to occur. May lead to less efficient resource utilization in order to guarantee no deadlocks.
 - Avoidance: divide into safe and unsafe state (Banker's algorithm). Each process declare max resource requirement for each resource type, and the system maintains necessary information to avoid unsafe state.

- Deadlock prevention: must find a way to eliminate one of the four necessary conditions for deadlock:
 - Don't allow exclusive access. This is probably not reasonable for many applications.
 - Create enough resources so that there's always plenty for all.
 - Don't allow waiting. (*phone company solution*) This punts the problem back to the user.
 - Allow preemption. *Preempt student's thesis disk space?*

- Attack hold & wait: Make process ask for everything at once. Either get them all or wait for

them all. Tricky to implement: must be able to wait on many things without locking anything. Problems?

- Attack circular wait: Define an ordering among resources and enforce it. Make *ordered* or *hierarchical* requests (e.g., ask for all S's, then all T's, etc.). All processes must follow the same ordering scheme. Problem?
- Integrated approach to deadlock: combine basic approaches, since none of the basic approaches alone is appropriate for entire spectrum of resource allocation in OS.
 - Partition resources into classes that are hierarchically ordered. -> resource ordering
 - Within each class, use one of the basic approaches.
- In general, prevention of deadlock is expensive and/or inefficient. Detection is also expensive and recovery is difficult (what if process has things in a weird state?).
- Is it possible to have a deadlock involving only 1 process?
- Does the existence of a cycle in the wait-for graph guarantee a deadlock?
- Is starvation different from deadlock?