V22.0202-001
Computer Systems Organization II (Honors)
(Introductory Operating Systems)

Lecture 10
Deadlocks

February 23, 2000

Outline

• Announcements
  – New TA for course: Lexing Ying (lexing@cs.nyu.edu)
    • office hours: 3:00 – 4:00pm Tuesdays, 4:00 – 5:00pm Thursdays
  – Write-up 2 due February 28th
  – Lab 3 due March 1st; start early!
  – Midterm review questions: I will discuss some of these on March 1st
  – Midterm exam on March 6th
  – Questions?

• Nachos labs: Group reorganization and dynamics

• Process deadlocks
  – system model
  – deadlock characterization
  – methods for handling deadlocks
  – deadlock prevention

Silberschatz/Galvin: Sections 7.1 - 7.4

Nachos Labs: Group Reorganization

• Current status: New groups

  Group 1: Akshay Arora, Abraham Chiswick, Jon Levy, Jordan Piel
  Group 2: Daniel Glick, Jesse Glick, Dipin Hora, Michael Allen
  Group 3: Eli Collins, Josh Harman, Yuri Niyazov, Mariano Belinky
  Group 4: Michael Allen, Mariano Belinky, Igor Faybyshev, Mikhail Radin
  Group 5: Jon Lorusso, Brian Seitz, Thomas Redis, Angelo Gagliano
  Group 6: Alex Beregovsky, Tugrul Galatali, Jesse Shaver, Igor Faybyshev
  Group 7: Bolei Guo, Junko Shirato, Vera Wong, Suzanna S. Yau
  Group 8: Michael Radin
  Group 9: Mikhail Radin

Nachos Labs: Group Dynamics

• All group members are expected to contribute to all of the labs
  – this means I expect you to design your solution as a team
  – even if you divide up the coding work,
    • every group members should have looked over/checked all of the code
  – I expect you to help clarify the material to each other
  – implications
    • everybody should be familiar with all aspects of the solution
    • “I did not work on that part of the Lab” is unacceptable
      • the entire group will be penalized for this

• Working in a group
  – is difficult: getting schedules to work out, different styles, etc.
  – but it is an important skill that will serve you well in the future

• Come talk to me if you feel that your group is not working out
Outline

* Announcements
  - New TA for course: Lexing Ying (lexing@cs.nyu.edu)
  - office hours: 3:00 – 4:00pm Tuesdays, 4:00 – 5:00pm Thursdays
  - Write-up 2 due February 28th
  - Lab 3 due March 1st; start early!
  - Midterm review questions: I will discuss some of these on March 1st
  - Midterm exam on March 6th
  - Questions?

* Nachos labs: Group reorganization and dynamics

* Process deadlocks
  - system model
  - deadlock characterization
  - methods for handling deadlocks
  - deadlock prevention

Silberschatz/Galvin: Sections 7.1 - 7.4

Process Deadlock

* Definition
  
  A deadlock occurs when a set of processes in a system is blocked waiting on requirements that can never be satisfied.

  These processes, while holding some resources, are requesting accesses to resources held by other processes in the same set.

  In other words, the processes are involved in a circular wait.

* Example:
  
  - 2 processes, each holding a different resource in exclusive mode, and each requesting access to the resource held by the other process
  
  - e.g., processes requiring access to disk and printer
  
  - one process acquires the disk and waits for the printer
  
  - the other acquires the printer and waits for the disk
  
  - neither will make progress!

  - resolving the deadlock requires the intervention of some process outside those involved in the deadlock

Deadlock versus Starvation

* Deadlock: A process waits for a resource that is currently assigned to another process, which is in turn waiting for another resource …

* Starvation: A process waits for a resource that continually becomes available but is never assigned to the waiting process

* Two major differences between deadlock and starvation
  
  - in starvation, it is not certain that a process will never get the requested resource (i.e., there is a chance it might), while a deadlocked process is permanently blocked
  
  - in starvation, the resource under contention is continuously available, whereas this is not true in a deadlock

* Starvation is typically easier to fix than deadlocks

System Model for Deadlocks

* Resources
  
  - different types of resources (e.g., memory space, CPU cycles, files)
  
  - processes request a resource type, not a particular resource
    
    - any of the resources in that type can be used to satisfy the request

* Processes
  
  - use resources
    
    - request resource type i
    
    - use resource type i
    
    - release resource type i

  - a process can request multiple instances of a resource
  
  - OS intervenes on request and release

* Deadlocks: Caused by processes waiting for events that never happen
  
  - events of interest: request and release
  
  - events can be for different resource types
Conditions for Deadlock

- Deadlocks involve a set of processes contending for a set of resources

- All of the following conditions must hold for deadlock to occur
  - Mutual Exclusion
    - at least one resource can only be used by one process at any one time
  - Hold and Wait
    - there must exist at least one process that is holding at least one resource, and is waiting to acquire additional resources currently held by other processes
  - No Preemption
    - processes cannot be forced to give up resources
  - Circular Wait
    - there is a sequence of processes $p_1, p_2, ..., p_n, p_1$, such that $p_i$ is waiting for a resource held by $p_{i+1}$

Graph Representations of Deadlocks

A Resource Allocation Graph (RAG)

- Two types of nodes
  - Processes and Resources

- Three types of directed edges between Processes and Resources
  - request edge: a solid edge from $P$ to $r$, indicating that $P$ has requested $r$
  - assignment edge: a solid edge from $r$ to $P$, indicating that the OS has already allotted resource $r$ to process $P$
  - claim edge: a dotted edge from a process node $P$ to a resource node $r$, indicating that $P$ may request $r$ at some point in the future

- We shall focus only on requests for exclusive access to a resource
  - handling of mixed access types is slightly complicated

Conditions for Deadlock: Dining Philosophers

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

  philosophers hold on to a chopstick while requesting another
  not possible to force a philosopher to give up a chopstick
  philosopher$_i$ waits on philosopher$_{i+1}$ ...

Resource Allocation Graphs: Example

- Assignment edges originate from a resource instance
  - A request edge is instantaneously transformed to an assignment edge if resources are available
Deadlocks in RAGs with Single Resource Instances

- A cycle in the graph is a necessary and sufficient condition for the existence of a deadlock.

Deadlocks in RAGs with Multiple Resource Instances

- A cycle in the graph is a necessary but not sufficient condition for the existence of a deadlock.
  - if a cycle does not exist: no deadlock
  - if a cycle exists: there may or may not be a deadlock

Methods for Handling Deadlocks

Three approaches with different cost-performance tradeoffs

- **Prevention**
  - deadlock cannot possibly occur

- **Avoidance**
  - deadlock can occur, but there are algorithms to avoid it
  - relies on the OS having an advance model of possible resource requests from processes

- **Detection and Recovery**
  - deadlock may occur, but there are ways of detecting it and recovering
  - this method is preferable when deadlocks happen rarely

Deadlock Prevention

- Approach: Ensure that the necessary conditions for deadlocks are never satisfied

- Prevent one of the following from becoming true
  - Mutual Exclusion
  - Hold and Wait
  - No Preemption
  - Circular Wait
Deadlock Prevention: Mutual Exclusion

- Mutual exclusion is not a problem for sharable resources
  - an example is a “read-only” file which is a resource that can be accessed simultaneously

- Problem: Some resources are inherently not sharable
  - so, denying the mutual exclusion condition cannot be enforced in general

Deadlock Prevention: Hold-and-Wait

- Approach: Guarantee that when a process requests a resource it does not hold any other resources
  - Choice 1: A process requests and is allocated all of its resources before it begins execution
    - require system calls requesting resources to precede all other system calls
  - Choice 2: A process releases any resources it is holding before it requests for new ones
  - Choice 3: A process that is “holding” a resource immediately releases it if another of its requests cannot be satisfied currently

- Limitations
  - inefficient
  - lowered resource utilization
  - starvation

Deadlock Prevention: No Preemption

- Approach: Take away resources from a process (preemption) and give them to another waiting process
  - some resources are preemptible
    - e.g., memory space, disk space (on a particular disk)
    - these can be taken away from a process
  - examples of non-preemptible resources?

- Choices
  - protocol 1: if a process is holding some resources and requests for other resources that cannot be granted to it, all its resources are taken away
  - protocol 2: when a process requests for additional resources, see if these resources are being held by a process that is itself waiting for new resources. In this situation, preempt the second process

- Limitation: Cost of preemption
  - a process may get preempted even when there is no deadlock

Deadlock Prevention: Circular Wait

- Approach: Order resource acquisition
  - all the N types of resources in the system are linearly ordered
    - each is given a number, called rank, in the range 1, 2, ... , N
    - the resources of the same type all have the same rank
    - different types of resources get distinct ranks
  - processes are required to sequence their resource requests in strictly increasing order of rank
    - i.e., they ask for all the “smaller” rank resources first

- Example:
  - three resources: memory, disk, printer
    - rank(memory) = 1, rank(disk) = 2, rank(printer) = 3
  - all processes
    - ask for memory first, then disk, then printer
  - is deadlock possible?
Deadlock Prevention: Circular Wait (contd.)

- Why it works
  - suppose the circular wait consists of processes P1, P2, ..., Pn, P1
  - suppose P(i) is waiting on a resource held by P(i+1) of rank R(i)
    - P(i+1) must have been granted all resources it needs of rank R(i)
    - it must therefore be waiting for a R(i+1) > R(i)
  - since a cycle of strictly increasing ranks cannot exist, there can exist no such cycle.

- Two related points
  - an equivalent strategy is one where a process, when it requests a resource of a particular rank, releases all those with a higher rank
  - typical rank orders are based on natural usage
    - e.g., since storage devices are used “before” printers, they get smaller ranks

Lecture Summary

- Process deadlocks
  - definition
  - system model
  - characterization
    - necessary conditions
    - resource allocation graph
  - deadlock prevention approaches

- Limitations of deadlock prevention
  - inefficient
    - static allocation of resources reduces concurrency
    - a process may need to be preempted even when there is no deadlock
  - restrictive
    - requires allocation of future resource requirements before it starts executing

Next Lecture

- Process Deadlocks (contd.)
  - deadlock avoidance
  - deadlock detection
  - deadlock recovery
  - combined approach to deadlock handling

Readings

- Silberschatz/Galvin: Sections 7.5 - 7.9