CS 370: OPERATING SYSTEMS
[DEADLOCKS]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Exponential Moving Average
  - Is the α adjusted? How is the first estimate (i.e. τ_0) chosen?
- MFQ
  - Any user-level control over setting the quanta? No
  - Must there be a quanta for every priority level? YES
  - Is the quanta in milliseconds or nanoseconds? MIlliseconds
  - At a given priority level what is the order of processes being scheduled?
  - When does a process get off the MFQ?
- HLT on x86 halts the CPU until the next external interrupt
- How does a thread first get assigned to a core?

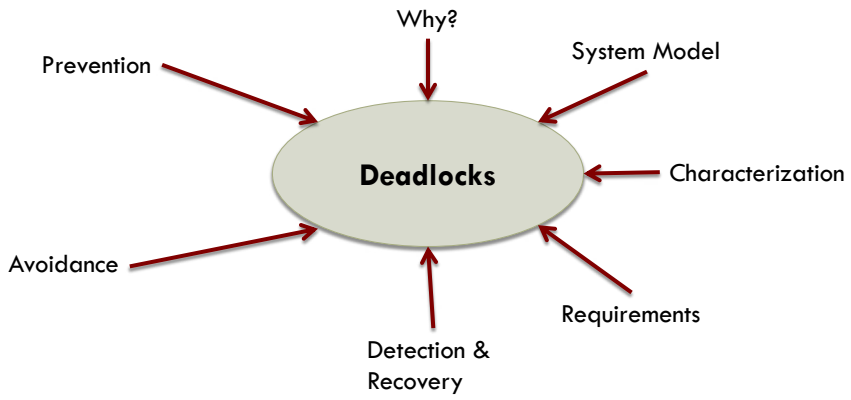
Topics covered in today's lecture

- Deadlocks
- Deadlock characterization
- Deadlock vs Starvation
- Resource allocation graph

A waiting process is never again able to change state
It is waiting for resources held by other processes

DEADLOCKS

What we will look at …



For many applications, processes need exclusive accesses to multiple resources

- Process A: Asks for scanner and is granted it
- Process B: Asks CD recorder first and is granted it.
- Process A: Now asks for CD recorder
- Process B: Now asks for Scanner

- Both processes are blocked and will remain so forever!
  - **Deadlock**

Other deadlock situations

- Distributed systems involving multiple machines
- Database systems
  - Process **1** locks record R1
  - Process **2** locks record R2
  - Then, processes **1** and **2** try to lock each other's record
    - Deadlock
- **Deadlocks can occur in hardware or software resources**

Resource Deadlocks

- Major class of deadlocks involves resources
  - Can occur when processes have been granted access to devices, data records, files, etc.
  - Other classes of deadlocks: communication deadlocks, two-phase locking
- Related concepts
  - Livelocks and starvation

Preemptable resources

- Can be taken away from process owning it with no ill effects
- Example: Memory
  - Process **B**'s memory can be taken away and given to process **A**
    - Swap **B** from memory, write contents to backing store, swap **A** in and let it use the memory

Non-preemptable resources

- Cannot be taken away from a process without causing the process to fail
- If a process has started to burn a CD
  - Taking the CD-recorder away from it and giving it to another process?
    - Garbled CD
    - CD recorders are not preemptable at an arbitrary moment
- In general, **deadlocks involve non-preemptable resources**

Some notes on deadlocks

- The OS typically does not provide deadlock prevention facilities
- Programmers are *responsible* for designing deadlock free programs

System model

- **Finite** number of resources
  - Distributed among *competing processes*
- Resources are *partitioned* into different ***types***
  - Each *type* has a number of identical instances
  - Resource type examples:
    - Memory space, files, I/O devices

A process must utilize resources in a sequence

- **Request**
  - Requesting resource must *wait until it can acquire* resource
  - `request()`, `open()`, `allocate()`
- **Use**
  - Operate on the resource
- **Release**
  - `release()`, `close()`, `free()`

For kernel managed resources, the OS maintains a system resource table

- Is the resource free?
  - Record process that the resource is allocated to
- Is the resource allocated?
  - Add to queue of processes waiting for resource
- For resources not managed by the OS
  - Use `wait()` and `signal()` on semaphores

Deadlock: 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.*
- Because all processes are waiting, none of them can cause events to wake any other member of the set
  - Processes continue to **wait forever**

DEADLOCK CHARACTERIZATION

Deadlocks: Necessary Conditions (I)

- **Mutual Exclusion**
  - At least one resource held in *nonsharable* mode
  - When a resource is being used
    - Another requesting process must wait for its release
- **Hold-and-wait**
  - A process must hold one resource
  - Wait to acquire additional resources
    - Which are currently held by other processes

Deadlocks: Necessary Conditions (II)

- **No preemption**
  - Resources cannot be preempted
  - Only voluntary release by process holding it
- **Circular wait**
  - A set of $\{P_0, P_1, \ldots, P_n\}$ waiting processes must exist
    - $P_0 \rightarrow P_1$; $P_1 \rightarrow P_2$, …, $P_n \rightarrow P_0$
  - Implies hold-and-wait

DEADLOCKS VS. STARVATION

Deadlocks vs. Starvation

- Deadlocks and starvation are both **liveness** concerns
- Starvation
  - Task fails to make progress for an indefinite period of time
- Deadlock is a *form of starvation*, BUT with a stronger condition
  - A group of tasks forms a **cycle** where none of the tasks makes progress because each task is waiting for some other task in the cycle to take action

Deadlocks vs. Starvation

- Deadlock implies starvation (literally for the dining philosophers problem)
- Starvation DOES NOT imply deadlock

Also …

- Just because a system can suffer deadlock or starvation does not mean that it always will
  - A system is *subject to starvation* if a task could starve in some circumstances
  - A system is *subject to deadlock* if a group of tasks could deadlock in some circumstances
- **Circumstances** impact whether a deadlock or starvation may occur
  - Choices made by scheduler, number of tasks, workload or sequence of requests, which tasks win races to acquire locks, order of task activations, etc.

RESOURCE ALLOCATION GRAPH

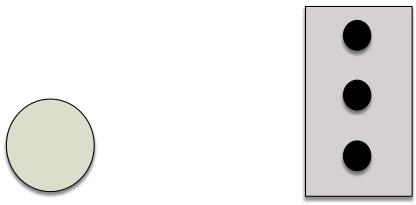
Resource allocation graph

- Used to describe deadlocks precisely
- Consists of a set of vertices and edges
- Two different sets of nodes
  - P: the set of all **active processes** in system
  - R: the set of all **resource types** in the system

Directed edges

- **Request** edge
  - P_i has requested an instance of resource type R_j
  - Directed edge from process P_i to resource R_j
  - Denoted $P_i \rightarrow R_j$
  - *Currently waiting* for that resource
- **Assignment** edge
  - Instance of resource R_j assigned to process P_i
  - Directed edge from resource R_j to process P_i
  - Denoted $R_j \rightarrow P_i$

Representation of Processes and Resources

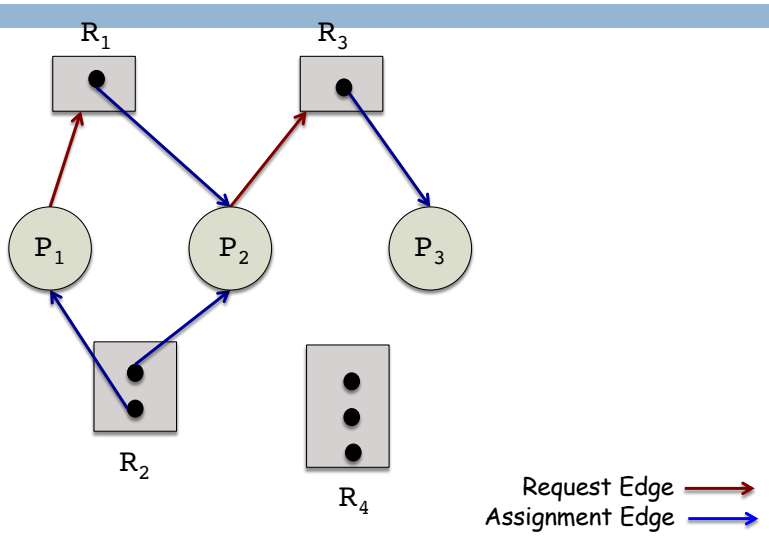


Processes

Resources

A resource type may have multiple instances

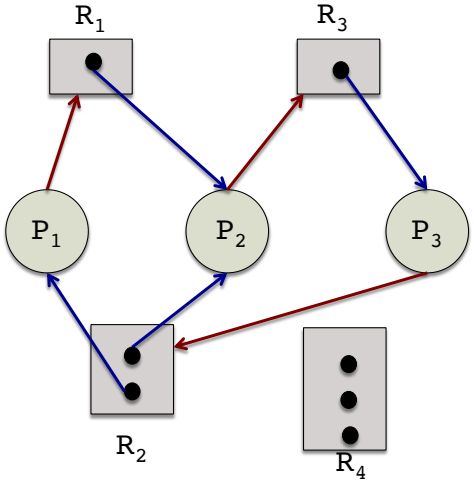
Resource Allocation Graph example



Determining deadlocks

- If the graph contains **no cycles**?
  - No process in the system is deadlocked
- If there is a **cycle** in the graph?
  - If each resource type has *exactly one* instance
    - Deadlock ***has*** occurred
  - If each resource type has *multiple* instances
    - A deadlock ***may have*** occurred

Resource Allocation Graph: Deadlock example

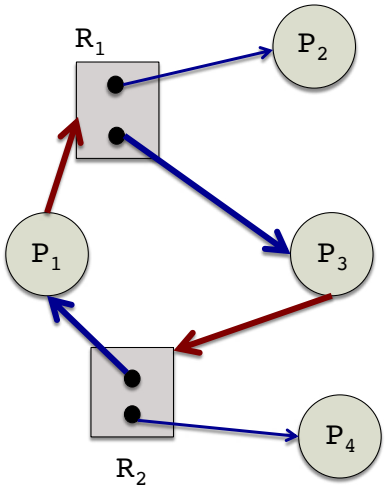


Two cycles

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Resource Allocation Graph: Cycle but not a deadlock



$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

P_4 may release instance of R_2 allocate to P_3 and break cycle

Resource Allocation Graphs and Deadlocks

- If the graph does not have a cycle
  - No deadlock
- If the graph does have a cycle
  - System may or may not be deadlocked

Methods for handling deadlocks

- Use protocol to **prevent** or **avoid** deadlocks
  - Ensure system never enters a deadlocked state
- Allow system to enter deadlocked state; BUT
  - **Detect** it and **recover**
- Ignore problem, pretend that deadlocks never occur

Problems with undetected deadlocks

- Resources held by processes that cannot run
- More and more processes enter deadlocked state
  - When they request more resources
- **Deterioration** in system performance
  - Requires restart

When is ignoring the problem viable?

- When they occur infrequently (once per year)
  - Ignoring is the *cheaper* solution
  - Prevention, avoidance, detection and recovery
    - Need to run constantly

SOME DEADLOCK EXAMPLES

Law passed by Kansas Legislature … early 20th Century

"*When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone*"

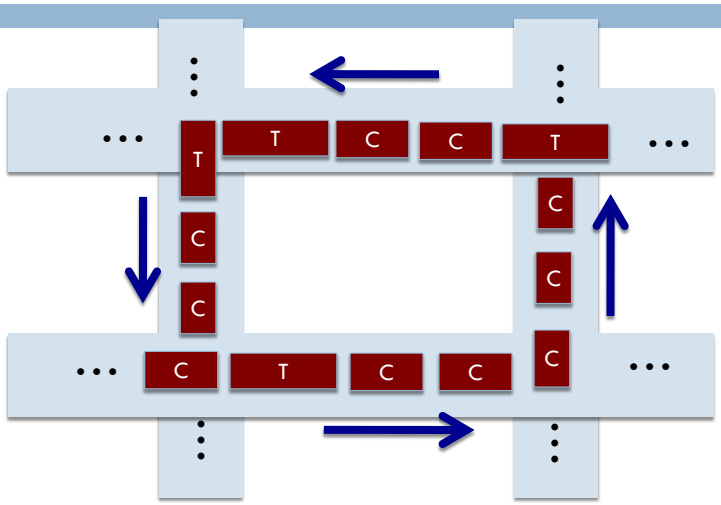
Dining philosophers problem: Necessary conditions for deadlock (1)

- Mutual exclusion
  - 2 philosophers *cannot share* the same chopstick
- Hold-and-wait
  - A philosopher *picks up one* chopstick at a time
  - Will not let go of the first while it *waits for the second* one

Dining philosophers problem: Necessary conditions for deadlock (2)

- No preemption
  - A philosopher *does not snatch chopsticks* held by some other philosopher
- Circular wait
  - Could happen if each philosopher *picks chopstick with the same hand* first

Is there a traffic deadlock here?



The traffic scenario: Necessary Conditions (1)

- Mutual Exclusion
  - A vehicle needs its *own space*
  - We can't stack automobiles on top of each other
- Hold-and-wait
  - A vehicle does not move and *stays in place* if it cannot advance

The traffic scenario: Necessary Conditions (2)

- No preemption
  - We *cannot move* an automobile to the side
- Circular-wait
  - Each vehicle is waiting for the one in front of it to advance

DEALING WITH DEADLOCKS

Four strategies for dealing with deadlocks

- Ignore the problem
  - May be if you ignore it, it will ignore you
- Detection and Recovery
  - Let deadlocks occur, detect them, and take action
- Deadlock avoidance
  - By careful resource allocation
- Deadlock prevention
  - By structurally negating one of the four required conditions

THE OSTRICH ALGORITHM

Ostrich Algorithm

- Stick your head in the sand; pretend there is no problem at all
- Reactions
  - Mathematician: Unacceptable; prevent at all costs
  - Engineers: How often? Costs? Etc.

OS suffer from deadlocks that are not even detected [1/3]

- Number of processes in the system
  - Total determined by slots in the process table
    - Slots are a finite resource
- Maximum number of open files
  - Restricted by size of the inode table
- Swap space on the disk

OS suffer from deadlocks that are not even detected [2/3]

- Every OS table represents a **finite** resource
- Should we abolish all of these because collection of n processes
  1. Might claim $1/n$ th of the total AND
  2. Then try to claim another one
- Most users prefer occasional deadlock to a restrictive policy
  - E.g. All users: 1 process, 1 open file …. one everything is far too restrictive

OS suffer from deadlocks that are not even detected [3/3]

- If deadlock elimination is free
  - No discussions
- But the price is often high
  - Inconvenient restrictions on processes
- Tradeoff
  - Between **convenience** and **correctness**

The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 5, 7]
- *Andrew S Tanenbaum. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620.* [Chapter 7]
- *Thomas Anderson and Michael Dahlin. Operating Systems Principles and Practice. 2nd Edition. ISBN: 978-0985673529.* [Chapter 6]