Chapter 6: Deadlocks

- Computer systems are full of resources that can be used only by one process at a time
  - printers, tape drives, ...
- All OS have the ability to grant a process exclusive access to certain resources
- Deadlock:
  - Example: two processes each want to record a scanned document on a Blu-ray disc.
  - process A requests permission to use the scanner and is granted it
  - process B is programmed differently and requests the Blue-ray recorder first and is also granted it.
  - Now A asks for the Blu-ray recorder, but the request is suspended until B releases it
  - Instead of releasing the Blue-ray recorder, B asks for the scanner
  - At this point both processes are blocked and will remain so forever
- can occur on hardware and software resources

Resources:

- devices, data records, files, and so forth
- a resource can be a hardware device or piece of info

Preemptable and Nonpreemptable Resources:

- Preemptable resource: is one that can be taken away from the process owning it with no ill effects
  - ex: memory
- Nonpreemptable resource: is one that cannot be taken away from its current owner without potentially causing failure
  - if a process has begun to burn a Blu-ray, suddenly taking the Blu-ray recorder away from it and giving it to another process will result in a garbled Blu-ray
- potential deadlocks that involve preempt able resources can usually be resolved by reallocating resources from one process to another
- a process using a resource will
  - 1. request the resource
    - if not available the process has to wait or sometimes blocked and awakened when the resource is ready
  - 2. use the resource
  - 3. release the resource

Resource Acquisition:

- for some resources, such as records in a database system, it is up to the user processes rather than the system to manage reduce usage themselves
    - one way is having a semaphore for each resource
        - to require the resource. use of the resource, and finally an up on the resource to release it

Introduction to Deadlocks:

- resource deadlock: each member of the set of deadlocked processes is waiting for a resource that is owned by a deadlocked process.

Conditions for Resource Deadlocks

- four conditions must hold for there to be a resource deadlock:
    - 1. Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
    - 2. Hold and wait condition. Processes currently holding resources that were granted earlier can request new resources.
    - 3. no-preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them
    - 4. Circular wait condition. There must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Deadlock Modeling:

- Four strategies are used fir dealing with deadlocks:
    - 1. Just ignore the problem. Maybe if you ignore it, it will ignore you
    - 2. detection and recovery. Let them occur, detect them, and take action
    - 3. dynamic avoidance by careful resource allocation
    - 4. prevention, by structurally negating one of the four conditions

The ostrich algorithm:

- stick your head in the sand and pretend there is no problem

Deadlock detection and recovery:

- the system does not attempt to prevent deadlocks from occurring, but instead it lets them occur, tries to detect when this happens, and then takes some action to recover

Deadlock detection with one resource of each type:

- algorithm:
    - 1. for each node, N, in the graph, perform the following 5 steps with N as the starting node
    - 2. initialize L to the empty list, and designate all the arcs as unmarked
    - 3. add the current node to the end of L and check to see if the node now appears in L two time. if it does, the graph contains a cycle and the algorithm terminates
    - 4. from the given node, see if there are any unmarked outgoings arcs. If so, go to step 5; if not got to step 6
    - 5. pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3
    - 6. if this node is the initial node, the graph does not contain any cycles and the algorithm terminates.Otherwise we have now reached a dead end. Remove it and go back to the previous node, that is, the one that was current just before this one, make that one the current node, and got to step 3.
- This algorithm takes each node , in turn, as the root of what it hopes will be a tree and does DFS on it
- If it ever comes back to a node it has already encountered, then it has found a cycle. If it exhausts all the arcs from any given node, it backtracks to the previous node. If it backtracks to the root and cannot go further, the subgraph reachable from the current node does not contain any cycles. If this property holds for all nodes, the entire graph is cycle free, so the system is not deadlocked.

Deadlock Detection with multiple resources of each type:

- algorithm:
    - 1. look for an unmarked process, Pi, for which the ith row of R is less than or equal to A
    - 2. if such a process is found, add the ith row of C to A, mark the process and go back to step 1
    - 3. if no such process exists, the algorithm terminates
- When the algorithm finishes, all the unmarked process, are deadlocked

Recovery through Preemption:

- possible to temporarily take a resource away from its current owner and give it to another process

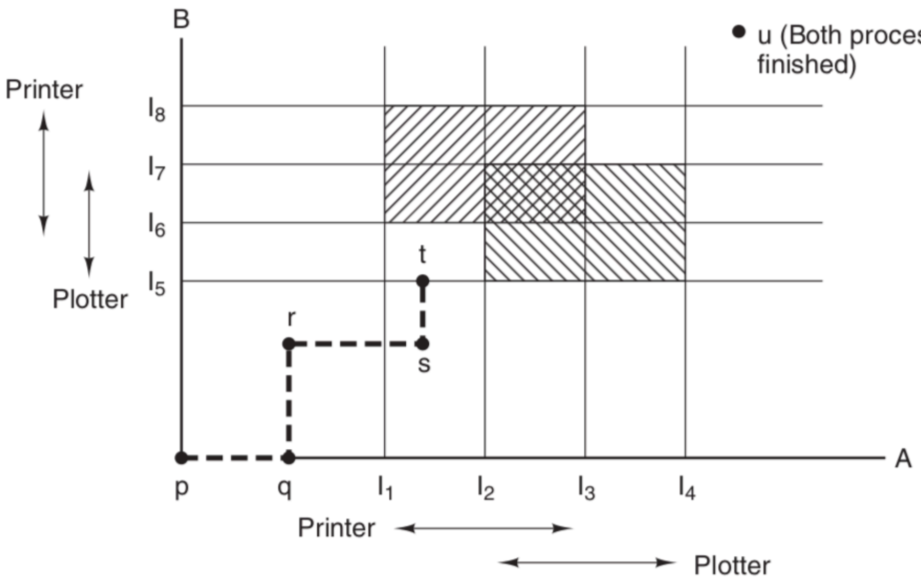
Recovery through Rollback

- if the system designers and machine operators know that deadlocks are likely, they can arrange to have process check pointed periodically
- checkpointing a process means that its state is written to a file so that it can be restarted later
- checkpoint contains the memory image and the resource state
- When a deadlock is detected, to do the recovery a process that owns a needed resource is rolled back to a point in time before it acquired that resource bus tarting at one of its earlier checkpoints

Recovery through killing processes

- kill a process in the cycle, and maybe the other process could run after
- a process not in the cycle can be chosen to kill if they have resources that some process in the cycle needs

Resource Trajectories:



-

Safe and Unsafe:

- a state is safe is there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately

- safe state: can guarantee that all processes will finish
- unsafe: no such guarantee can be given

Banker Algorithm :

- scheduling algorithm that can avoid deadlocks
- algorithm checks to see if granting the request leads to an unsafe state, if so the request is denied.

Deadlock Prevention:

Attacking the Mutual-Exclusion Conditions:

- if no resource were assigned exclusively to a single process

Attacking the hold and wait condition

- if we can prevent that hold resources from waiting for more resources, we can eliminate deadlocks
- one way is to require all processes to request all their resources before starting execution
- Or require a process requesting a resource to first temporarily release all the resources it currently holds. Then it tries to get everythignit needs all at once

Attacking the Non-Preemption Condition

- some resources can be virtualized to avoid no preemption condition

Attacking the circular wait condition

- one way is simply to have a rule saying that a process is entitled only to a single resource at any moment
- second way is to provide a global numbering of all the resources
    - a process can request resources whenever they want to, but all requests must be made in numerical order

Two-Phase Locking:

- first phase : the process tries to lock all the records it needs, one at a time. if it succeeds, it begins the second phase

- second phase: performing its updates and releasing locks

Communication Deadlocks::

- two or more processes communicate by sending messages
- A common arrangement is that process *A* sends a request message to process *B*, and then blocks until *B* sends back a reply message. Suppose that the request message gets lost. *A* is blocked waiting for the reply. *B* is blocked waiting for a request asking it to do something. We have a deadlock
- But it is a deadlock according to our formal definition since we have a set of (two) processes, each blocked waiting for an event only the other one can cause. This situation is called a **communication deadlock**
-
-

Starvation

- some processes never getting service even though they are not deadlocked
- can be avoided by using first come first served