7 – Deadlocks

7 – Deadlocks

- What are deadlocks ?
- How to deal with deadlocks

What we've learned so far ...

- We want to utilize resources efficiently
  - Multiprogramming, time-sharing, etc.
  - At some scale, processes are units of work that can run in parallel.
    - Then, in finer grains, threads.
- Two or more processes (or threads, fibers) may need to access the same resource at the same time.
  - To be consistent, it may need to execute atomically.
  - Semaphores, Mutexes, ...

(cont'd.)

- But, further more, each process may need to access several resources (exclusively) to accomplish a task.
  - That introduces another problem, e.g.,
    - P_1 and P_2 may access both R_1 and R_2
    - P_1 accesses R_1 exclusively
    - P_2 accesses R_2 exclusively
    - P_1 must wait P_2 to release R_2 to accomplish its task
    - P_2 must wait P_1 to release R_1 to accomplish its task
    - Both P_1 and P_2 cannot go anywhere – **deadlock !**

What is it, formally ?

- A set of processes is deadlock if each process in the set is waiting for an event that only another process in the set can cause.

Conditions for Deadlock

- Defined by E. G. Coffman in 1971
- The following four conditions **must** hold:
  - Mutual Exclusion
    - Require to access a resource exclusively
  - Hold and Wait
    - Hold resources and wait for accessing other resources
  - No Preemption
    - The process must release the resource so that the others can access it.
  - Circular Wait
    - Two or more processes form a circular chain where one process is waiting for the resource that the next process in the chain holds.

How to deal with deadlocks ?

- First question, do we really need to prevent, avoid, or recover deadlocks ?
  - Mathematicians say "*Yes, we must*".
  - Engineers say "*That depends on how often and how serious*".
    - If deadlock << system crashes, hardware failures, bugs, then we should focus on solving those problems, and just ignore the deadlock.
- So, the first algorithm: the Ostrich algorithm – close your eyes, and pretend that there is no (deadlock) problem at all.
  - Interestingly, most UNIX systems, and MS Windows just ignore deadlocks.

(cont'd.)

- And to deal with deadlocks, we may use following strategies:
  - Deadlock Prevention
  - Deadlock Avoidance
  - Deadlock Detection and Recovery

Deadlock Prevention

- The strategy is to break the deadlock conditions.
- Mutual Exclusion
  - Virtually impossible to deny mutual exclusion to prevent deadlocks, some resources are not sharable.
  - Spooling may help.
- Hold and Wait
  - Request all resources before execution.
    - Low utilization
  - Release all resources it holds, and then request all.
- No Preemption
  - It is difficult to make some resources preemptible.

(cont'd.)

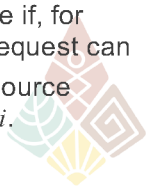
- Circular Wait
  - Total ordering resources, and let processes to request resources in increasing order.
  - To request a resource, the process must release all the lower-order resources.

Deadlock Avoidance

- The system must be able to decide whether granting a resource is safe or not and make allocation only when it is safe.
- Safe state
- Banker's algorithm

Safe State

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.
- Formally, a system is in a safe state *iff* there exists a *safe sequence*.
- A sequence of processes $< P_0, P_1, ..., P_n >$ is a safe sequence for the current allocation state if, for each P_i, the resources that P_i can still request can be satisfied by the current available resource pluses resources held by all P_j with $j < i$.



(cont'd.)

- Example:
  Total resource = 12

Process	Max needs	Need at t_0
P_0	10	5
P_1	4	2
P_2	9	2

- At t_0 the system is in a safe state
  - The sequence $< P_1, P_0, P_2 >$ satisfies the conditions, so $\{5, 2, 2\}$ can be granted.

(cont'd.)

- P_1 requests all resources to finish its job, thus $t_0 = \{5, 4, 2\}$ and only one resource available.
- P_1 finishes the job, and releases all holding resources $t_0 = \{5, 0, 2\}$, and 5 resources are available.
- Next, P_0 requests $t_0 = \{10, 0, 2\}$, then release $t_0 = \{0, 0, 2\}$.
- Finally, P_2, request $t_0 = \{0, 0, 9\}$, then release $t_0 = \{0, 0, 0\}$.
- A safe system may go unsafe
  - e.g., if $t_0 = \{5, 2, 3\}$.

Banker's Algorithm

- An algorithm to determine whether there is a safe state.
- Named after a real-world situation:
  - Customers have a credit limit. Bank cannot allow everyone to maximize their credits at once.
  - Banker has to queue customers in some sequences to satisfy all of them.
  - Customer = Process, Cash = Resource, Banker = OS

(cont'd)

- Let
  - m = no. of classes of resources
  - n = no. of processes
  - $A\,[n \times m]$ represents resources allocated by each process
    - One row per process
    - One column per class of resource
  - $N\,[n \times m]$ represents resources needed
  - $E\,[1 \times m]$ represents no. of resources of each class
  - $P\,[1 \times m]$ sums columns of A.

(cont'd.)

1. Decrease $N_{p,\,C \in m}$ and increase $A_{p,\,C \in m}$ according to the request. Compute P.
2. Select p' such that all elements of $N_{p'}$ are less than or equal to $\mathrm{E} - \mathrm{P}$. If no such p' exists, it is unsafe. Terminate algorithm, restore N, A, P.
3. Subtract $A_{p'}$ from P. Strike p' from further consideration.
4. Repeat step 2 – 3 until all rows in N have been processed.
5. If unsafe state does not exist, then it is safe and the request at step 1 can be committed.

(cont'd.)

- Example: 1 printer, 1 disk, 2 processes
  Initially,

$$A=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\quad N=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\quad E=[1\quad 1]\quad P=[0\quad 0]$$

  Suppose x request for a printer:

$$A=\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\quad N=\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\quad E=[1\quad 1]\quad P=[1\quad 0]$$

$$E-P=[0\quad 1]$$

  Since $N_x \le E - P$, continue

(cont'd.)

- Subtract A_x from P

$$A=\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\quad N=\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\quad E=[1\quad 1]\quad P=[0\quad 0]$$

$$E-P=[1\quad 1]$$

Since $N_y \leq E-P$, process y can continue.

- So, it is safe to allow x to allocates a printer.

(cont'd.)

- Next, if y request for a disk

$$A=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\quad N=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\quad E=[1\quad 1]\quad P=[1\quad 1]$$

$$E-P=[0\quad 0]$$

No $N_{p''} \leq E-P$. So, it is unsafe, and the request must be blocked.

(cont'd.)

- Example: is this safe ?

$$A=\begin{bmatrix}1 & 1 & 0\\ 0 & 0 & 0\\ 2 & 0 & 0\end{bmatrix}\quad N=\begin{bmatrix}0 & 1 & 0\\ 2 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}\quad E=[3\quad 2\quad 1]\quad P=[3\quad 1\quad 0]$$

Let's see,

Row Selected	P	$E-P$
	$[3\quad 1\quad 0]$	$[0\quad 1\quad 1]$
1	$[2\quad 0\quad 0]$	$[1\quad 2\quad 1]$
3	$[0\quad 0\quad 0]$	$[3\quad 2\quad 1]$
2	$[0\quad 0\quad 0]$	$[3\quad 2\quad 1]$

So, it is safe, and a safe sequence is $< 1, 3, 2 >$.

(cont'd.)

- Example: is this also safe ?

$$A=\begin{bmatrix}1&1&0\\0&0&1\\2&0&0\end{bmatrix}\quad N=\begin{bmatrix}0&1&0\\2&0&1\\1&1&1\end{bmatrix}\quad E=[3\quad 2\quad 1]\quad P=[3\quad 1\quad 1]$$

Row Selected	P	$E - P$
	[3 1 1]	[0 1 0]
1	[2 0 1]	[1 2 0]
??		

So, it is unsafe.

Deadlock Detection and Recovery

- Let deadlocks occur, and try to detect them.
  - And, if it is possible, recover them.
- If the resource-allocation graph contains at least one cycle, then deadlocks exist.
  - So, we can detect deadlocks by traverse through the graph to find whether any cycle exists.

Resource-Allocation Graphs

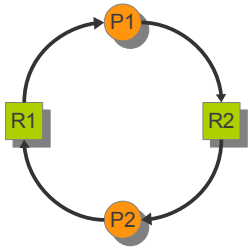
- In 1972, R. C. Holt showed how to represent resource allocation model using directed graphs.

R is held by P

P requests for R

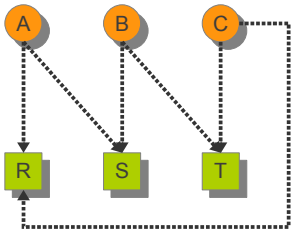
(cont'd.)

- This can be a deadlock:

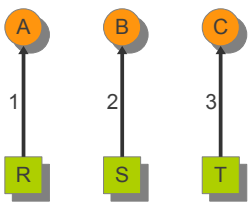


(cont'd.)

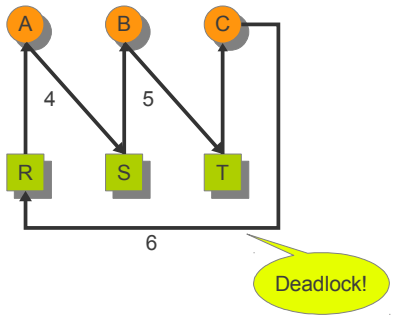
- The order of execution may or may not cause the deadlock, e.g., 3 processes access 2 of 3 resources, with round-robin, they might be scheduled to:



(cont'd.)

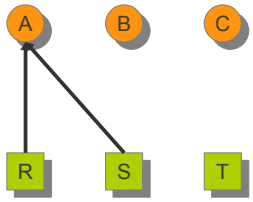


(cont'd.)

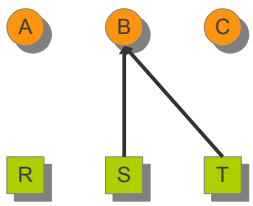


(cont'd.)

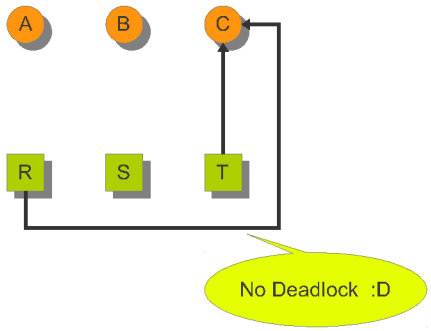
- If they are scheduled differently, e.g.,



(cont'd.)



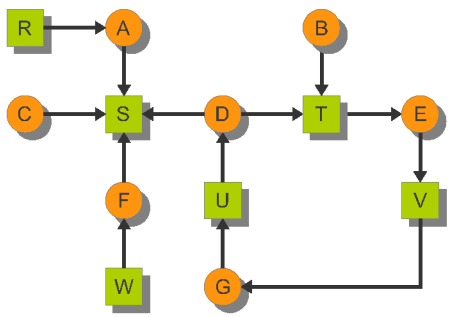
(cont'd.)



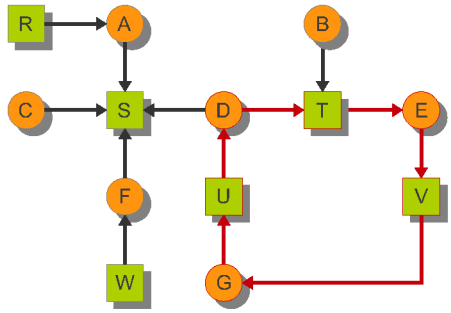
An example

- A holds R and wants S.
- B holds nothing but wants T.
- C holds nothing but wants S.
- D holds U and wants S and T.
- E holds T and wants V.
- F holds W and wants S.
- G holds V and wants U.

(cont'd.)



(cont'd.)



When to detect deadlocks ?

- Every time a resource request is made.
  - Quite expensive in term of CPU time
- Every certain period of time
- When CPU utilization has dropped below some threshold.
  - When deadlocks has occurred, there will be few runnable processes, and CPU will often be idle.

Deadlock Recovery

- Preemption
  - Temporarily take resource away from current process and assign to another.
  - Manual intervention may be required.
  - Difficult or even impossible to preempt
- Rollback
  - Back to the known consistent state – checkpoint
  - Imply implementation of checkpoint
    - Memory image
    - Resource states
    - Incremental checkpoint

(cont'd.)

- Destroy
  - Simplest way to recover,
  - Kill a process or processes to break the cycle
    - Randomly kill the processes in the cycle
    - Carefully chosen not-in-the-cycle processes that hold resources required by processes in the cycle.