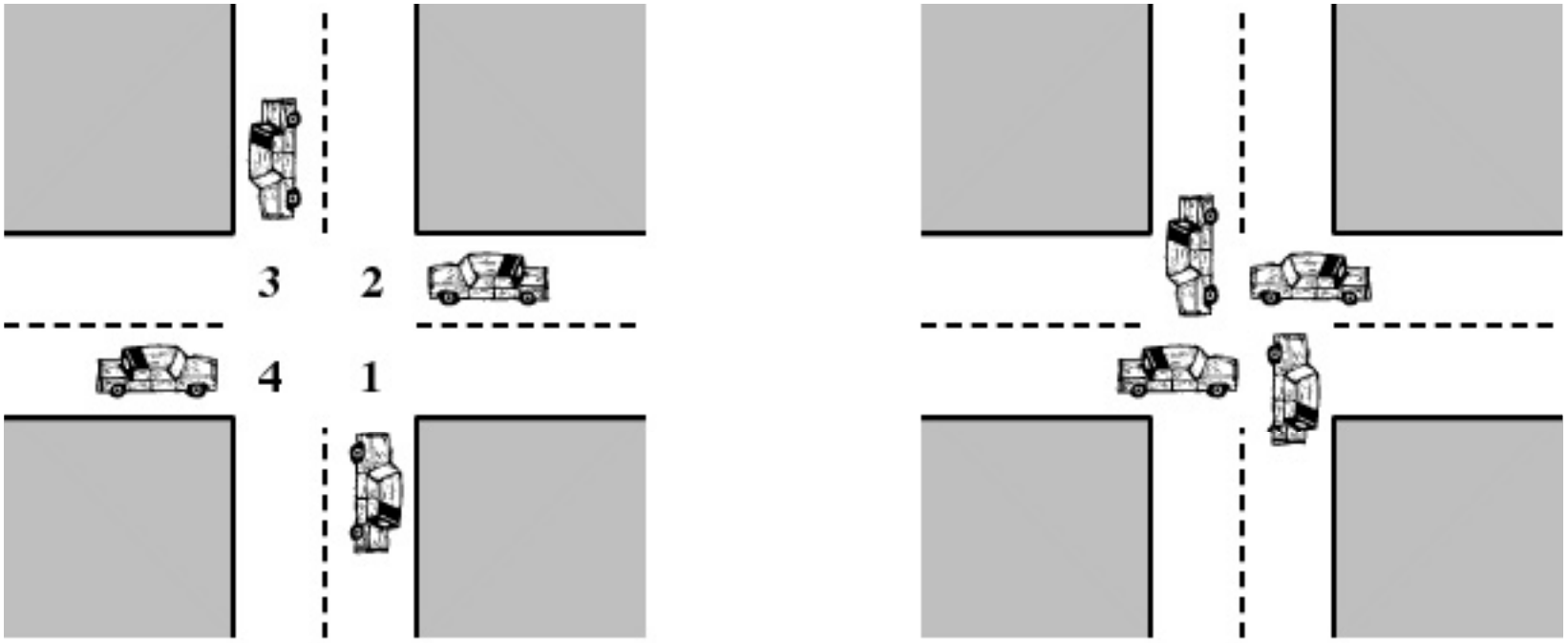
UNIVERSITI TENAGA NASIONAL

Concurrency: Deadlock and Starvation

CSNB153 COMPUTER SYSTEM

Deadlock

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other
- No efficient solution in general case
- Involve conflicting needs for resources by two or more processes



(a) Deadlock possible

(b) Deadlock

Figure 6.1 Illustration of Deadlock

Example Deadlock

- Consider two processes that compete for exclusive access to disk file D and tape drive T.
- The program operations shown in figure 6.4 below.

Process P

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q-

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

Conditions for Deadlock

- Four conditions of policy must be present for a deadlock to be possible:
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait

Conditions for Deadlock

1. Mutual exclusion

   Only one process at a time can use a resource

2. Hold-and-wait

   A process holding at least one resource and waiting to acquire additional resources held by other processes.

Conditions for Deadlock

3. No preemption

   A resource can be released only voluntarily by the process holding it after that process has completed its task.

4. Circular Wait

   A closed chain of processes exists, such that each process holds at least once resources needed by the next process in the chain. P0 is waiting for P1, P1 is waiting for a resource that is held by Pn and Pn is waiting for resource that is held by P0

Approaches for Deadlock

- The most important approaches that have been developed:
  1. Deadlock Prevention
     - Prevent deadlock by adopting a policy that eliminates one of the conditions
  2. Deadlock Avoidance
     - Avoid deadlock by making the appropriate dynamic choices based on the current state of resource allocation.
  3. Deadlock Detection
     - Attempt to detect the presence of deadlock and take action to recover.

Deadlock Prevention

- To design a system in such a way that the possibility of deadlock is excluded.
- Method falling into two classes:

1. Indirect method

   – to prevent the occurrence of one of the three necessary condition listed previously.

2. Direct method

   – to prevent the occurrence of circular wait.

Deadlock Prevention…

Techniques related to each four:

1. Mutual Exclusion (ME)
   - Cannot be disallowed → MUST BE ALLOWED!!
   - If access to resources require ME, then ME must be supported by the OS.
   - But deadlock still can occur (in some resources):

     E.g.. Files may allow multiple accesses for reads but only exclusive for writes.
     - Deadlock can occur if more than one process requires write permission.

Deadlock Prevention…

2. Hold and Wait

- Prevented by requiring that a process request all of its required resources at one time and blocked the process until all request can be granted simultaneously.
- This approach inefficient because:

  i. Process may be held up for along time waiting for all of its resources to be filled, when in fact it could have been proceeded with only some of the resources.

  ii. Resource allocated to a process may remain unused for a considerable period. During which time they are denied to other processes.

Deadlock Prevention…

iii. Process may not know in advance all of the resources that it will require.

Deadlock Prevention…

3. No Preemption

- Can be prevented in several ways:

  i. If a process holding certain resources is denied a further request, that process must release its original resources and if necessary request again with the additional resource.

  ii. If a process request a resource that is currently held by another process, the OS must preempt the second process and require it to release its resource.

  – Practical only when applied to resources whose state can be easily saved and restored later, e.g. a processor.

Deadlock Prevention…

4. Circular Wait

- Can be prevented by defining a linear ordering resource types.

  – If a process has been allocated resources of type *R*, then it may subsequently request only those resources of type following *R* in the ordering.

  - E.g.

    Let us associate an index with a each resource type. Then resource *Ra* precedes *Rc* in the ordering if *a< c (using alphabetical order)*.

Deadlock Prevention...

– Suppose that two processes, A and B, are deadlocked because A has acquired *Ra* and requested *Rc*, and B acquired *Rc* and requested *Ra*. This condition is impossible because it implies $a < c$ and $c < a$

– circular wait prevention may be inefficient because it will slowing down process and denying resource access unnecessarily.

Deadlock Avoidance

- Allow the 3 policy conditions but make judicious choices to assure that the deadlock point is never reached.
- Allows more concurrency than prevention.
- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process request

Deadlock Avoidance…

- Two approaches of deadlock avoidance:
  1. Do not start a process if its demands might lead to deadlock
  2. Do not grant an incremental resource request to a process if this allocation might lead to deadlock.
- In both cases: maximum requirements of each resource must be stated in advance.

Deadlock Avoidance…

Advantage:

- Does not necessarily preempt and rollback processes compared to deadlock detection.
- But has a number of restrictions:
  - Maximum resource requirement must be stated in advance
- Processes under consideration must be independent; no synchronization requirements.
- There must be a fixed number of resources to allocate
- No process may exit while holding resources

Deadlock Detection

- Do not limit resource access or restrict process actions.
- Resource access are granted to process whenever possible.
- OS will periodically performs:
  - an algorithm to check if deadlock present
  - an algorithm to recover from deadlock

Deadlock Detection…

- The deadlock check can be performed at every resource request.
- Checking at each resource request has two advantages:

  i. Leads to early detection

  ii. Algorithm is simple

  – because it is based on incremental changes to the state of the system.

- But such frequent checks will consume CPU time.

Deadlock Detection Recovery

- Needed when deadlock is detected.
- The following approaches are possible:

  1. Abort all deadlocked processes (one of the most common solution adopted in OS).

  2. Back up each deadlocked process to some previously defined checkpoint, and restart all process

Deadlock Detection Recovery

3. Successively abort deadlock processes until deadlock no longer exists

- After each abortion, need to re-invoke the deadlock detection algorithm to see either deadlock still exists or not.

4. Successively preempt some resources from processes and give them to other processes until deadlock no longer exists

- a process that has a resource preempted must be rolled back prior to its acquisition of that resource.

Deadlock Detection Recovery

- For approaches 3 and 4: a victim process needs to be selected according to:
  - Least amount of processor time consumed so far
  - Least number of lines of output produced so far
  - Most estimated time remaining
  - Least total resources allocated so far
  - Lowest priority

Integrated Deadlock Strategy

- Combine the previous approaches into the following way:
  - Group resources into a number of different resource classes.
  - Use the linear ordering strategy (prevention of circular wait) to prevent deadlock between resource classes
  - Within a resource class, use the algorithm that is most appropriate for that class.

Integrated Deadlock Strategy

- Example of resource classes:
  - Swappable space
    - Blocks of memory on secondary storage for use in swapping processes.
  - Process resources
    - Assignable devices, such as tape drives and files
  - Main memory
    - Assignable to processes in pages and segments
  - Internal resources
    - Such as I/O channels

Starvation

- is the name given to the indefinite postponement of a process because it requires some resource before it can run, but the resource, though available for allocation, is never allocated to this process.

CAUSES OF STARVATION

Starvation is caused by failure to allocate some resource to a process, so to find the causes we must inspect the policies which the system uses in handling resources. Here are some possibilities.

- Processes hand on resources to other processes without control. If processes queue for a resource, and the resource is always handed on to the next process in the queue, it is essential that every process awaiting the resource must be placed in the queue.
- Processes' priorities are strictly enforced. If a process of worse priority requires a resource in competition with a constant stream of processes of better priority, it might wait for ever.

- "Random" selection is used. If processes awaiting service are not queued, but a random process is selected whenever the resource becomes available, it is possible for some processes to wait for a very long time.
- Not enough resources. This is commonly the real problem, so far as physical resources are concerned, though as its solution costs money it might be a hard one to solve.

REMEDIES.

- There must be an independent manager for each resource, which must manage all allocations of its resource; this will guarantee that processes don't just pass resources around between themselves without making them available for general allocation.
- Strict priorities should not be enforced. A poor priority should be regarded as a weak claim, but not an over ridable claim.
- Avoid random selections, uncontrolled competition, etc.
- Provide more resources. This is the only satisfactory solution.

END.....