Deadlocks

- Non-sharable resource: the resource can be used by only one process at a time

- A process may use a resource in only the following sequence:
  - Request: If the resource cannot be granted, the requesting process must wait until it can acquire the resource.
  - Use: The process can operate on the resource.
  - Release: The process releases the resource.

- Deadlock:

Multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar state.
- Necessary conditions for a deadlock to occur

1. Mutual exclusion: The resource is non-sharable.

2. Hold and wait: A process that is holding resources can request new resources.

3. No preemption: A resource can be released only by the process holding it.

4. Circular wait: There is a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

5. All four conditions must simultaneously hold.
- Resource graph
  - Directed graph
  - Two kinds of nodes:

  Processes (circles) and resources (squares)
  - Arcs:

  Process P holds resource R

  Process P is waiting for resource R

  A cycle in the directed graph means there is a deadlock:

  Process P holds T and requests R

  Process Q holds R and requests T
- Use resource graph to detect deadlocks.

An example:

* Three processes A, B, and C
* Three resources R, S and T
* Round robin scheduling

- Using resource graph, we can see if a given request/release sequence leads to deadlock:

  Carry out the request and release step by step, check if there is any circle after each step.
• Methods for handling deadlocks

  1. Ignore the problem

  2. Detection and recovery

  3. Prevention

  4. Dynamic avoidance

• The Ostrich algorithm:

  – Stick your head in the sand and pretend that deadlocks never occur.

  – Used by most operating systems, including UNIX.

  – Tradeoff between convenience and correctness
• An example of deadlock in UNIX:

  – Process table has 100 slots

  – 10 processes are running

  – Each process needs to fork 12 subprocesses

  – After each forks 9 subprocesses, the table is full

  – Each original process sits in the endless loop: fork and fail
 Detection and recovery

In a system where a deadlock may occur, the system must provide:

- An algorithm than exams the state of the system to determine whether a deadlock has occurred
- An algorithm to recover from the deadlock

- Detection:

Every time a resource is requested or released, check resource graph to see if any cycles exist.

- How to detect cycles in a directed graph?

Depth-first search from each node. See if any repeated node. \( O(N) \) algorithm.
– Recovery:

* Abort one process at a time until the deadlock cycle is eliminated.

* A simpler way (used in large main frame computers):

  Do not maintain a resource graph. Only periodically check to see if there are any processes that have been blocked for a certain amount of time, say, 1 hour. Then kill such processes.

* To recover the killed processes, need to restore any modified files. Keep different versions of the file.
● Deadlock prevention:

Use a protocol to ensure that the system will never enter a deadlock state.

Negating one of the four necessary conditions.

1. Mutual exclusion

2. Hold and wait

3. No preemption

4. Circular wait
1. Mutual exclusion

- Ensure that no resource is assigned exclusively to a single process. Spooling everything.

- Drawback: not all resources can be spooled (such as process table)

- Competition for disk space for spooling itself may lead to deadlock.

2. Hold and wait

- Process requires all its resources before starting

- Problem: processes may not know how many resources needed in advance; not an optimal approach using resources (low utilization)
• A variant: a process requesting a resource first temporarily release all the resources it holds. Once the request is granted, it gets all resource back.

3. No preemption

Forcibly take away the resource. Not realistic.

4. Circular wait

• Solution 1: A process is entitled only a single resource at any time.

• Solution 2: Global numbering all resources:
  – Give a unique number to each resource
  – All requests must be made in numerical order
- An example: two processes and five devices. Number the resources as follows:
(a) Card reader
(b) Printer
(c) Plotter
(d) Tape drive
(e) Card punch
Assume process A holds $i$ and process B holds $j$ ($i \neq j$).
If $i > j$, A is not allowed to request $j$.
If $i < j$, B is not allowed to request $i$.

- Suitable to multiple processes. At any time, there must be a assigned resource with the highest number. This process will not request other assigned resources, only requests higher numbered resource and finishes. Then releases all resources.
• Deadlock avoidance

  - Analyzing each resource request to see if it can be safely granted.

  - Find a general algorithm that can always avoid deadlock by making right choice.

  - Banker’s algorithm for a single resource:

    * A small town banker deals with a group of customers with granted credit lines.

    * The analogy:
      Customers: processes
      Units: copies of the resource
      Banker: O.S.

    * State of the system:
      showing the money loaned and the maximum credit available
* Safe state:
  there exists a sequence of other states
  that lead to all customers getting loans
  up to their credit lines.

* Algorithm:

  For each request, see if granting it
  leads to a safe state. If it does, the
  request is granted. Otherwise, it is
  postponed until later.

Check a safe state:

(1) See if available resources can sats-
ify the customer closest to his max-
imum. If so, these loans are assumed
to be repaid.
(2) Then check the customer now closet
to his maximum, and so on.
(3) If all loans can be eventually paid,
the current state is safe.
- Resource trajectories:
  A model for two processes and two resources

* An example:
  Process A and B
  Resources: printer and plotter
  A needs printer from \( I_1 \) to \( I_3 \)
  A needs plotter from \( I_2 \) to \( I_4 \)
  B needs plotter from \( I_5 \) to \( I_7 \)
  B needs printer from \( I_6 \) to \( I_8 \)

* Each point in the diagram is a joint state of A & B

* Can only go vertical or horizontal (one CPU)

* Start at point p
  Run A to point q
  Run B to point r
  Run A to point s, granted printer
  Run B to point t, request plotter
  Can only run A to completion.
– Banker’s algorithm for multiple resources

* Processes must state their total resource needs before executing

* $n$ processes and $m$ types of resources

* Two matrices:

  Current allocation matrix  
  Request matrix

* Three vectors:

  Existing resource: $E = (E_1, E_2, \ldots, E_m)$  
  Possessed resource: $P = (P_1, P_2, \ldots, P_m)$  
  Available resource: $A = (A_1, A_2, \ldots, A_m)$  

$$A = E - P$$
* Algorithm:

(1) Look for a row R whose unmet resource needs $\leq A$. If no such row exists, the system will deadlock.

(2) Assume the process of row R requests all the resources it needs and finishes. Mark that process terminated and add all its resources to vector $A$.

(3) Repeat Step 1 and 2 until either all processes are marked terminated (the initial state is safe) or a deadlock occurs (the initial state is unsafe).
An example.

Row D \leq A, then \quad A = A + (1101) = (2121)
Row A \leq A, then \quad A = A + (3011) = (5132)
Row B \leq A, then \quad A = A + (0100) = (5232)
Row C \leq A, then \quad A = A + (1110) = (6342)
Row E \leq A, then \quad A = A + (0000) = (6342) = E

So, the current state is safe.

Suppose process B requests a printer

Now \quad A = (1010)
Row D \leq A, then \quad A = A + (1101) = (2111)
Row A \leq A, then \quad A = A + (3011) = (5122)
Row B \leq A, then \quad A = A + (0110) = (5232)
Row C \leq A, then \quad A = A + (1110) = (6342)
Row E \leq A, then \quad A = A + (0000) = (6342) = E

So, the request is still safe.

If E requests the last printer.

A = (1000)
No row \leq A, will lead to a deadlock.
So E's request should be deferred.
– Problems:

(1) Process don’t know the maximum resources they need in advance

(2) The number of processes is not fixed

(3) Available resources may suddenly break

• In summary,

Prevention: too overly restrictive

Avoidance: required information may not be available

Still no good general solution yet.