Final

Chapter-08

'Deadlock'

*** **Definition:** Deadlock is a situation where two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes. Deadlocks commonly occur in systems, databases, or multithreaded applications.

*** ☐ System Model:

System consists of Resources $(R_1, R_2, \ldots, R_m)$

Each process utilizes resource as follows:

- request
- use
- release

☐ Deadlock with Semaphores:

A semaphor S_1 initialized to 1

A semaphor S_2 initialized to 1

Two threads T_1 and T_2

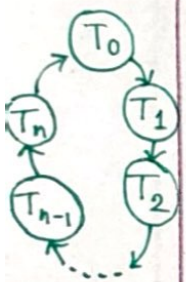
T_1 : wait (S_1)
wait (S_2)

T_2 : wait (S_2)
wait (S_1)

Deadlock Characterization

Deadlock can arise for 4 conditions hold simultaneously

- Mutual exclusion : only one thread at a time can use a resource.
- Hold and wait: A thread holding at least one resource is waiting to acquire additional resources held by other threads.
- No preemption: A resource can be released only voluntarily by the thread holding it, after the thread has completed its task.
- Circular wait: There exists a set $\{T_0, T_1, \ldots, T_n\}$ of waiting threads such that T_0 is waiting for a resource that is held by $T_2, \ldots, T_{n-1}$ is waiting for a resource that is held by T_n, and T_n is waiting for a resource that is held by T_0. Single instance resources are sufficient condition for deadlock.



circular wait

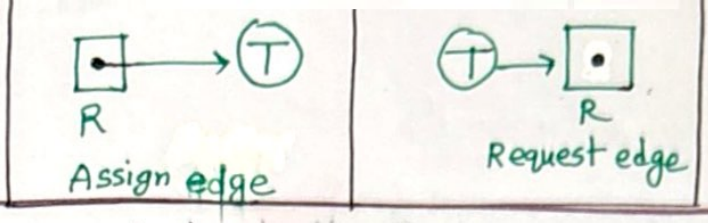
Resource Allocation graph

A set vertices V → V is partitioned into 2 types:

A set of edges E

- $T = \{T_1, T_2, \ldots, T_n\}$, the set consisting of all threads in the system.
- $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system

- requested edge – directed edge $T_i \rightarrow R_j$
- assignment edge – $R_j \rightarrow T_i$ directed edge.



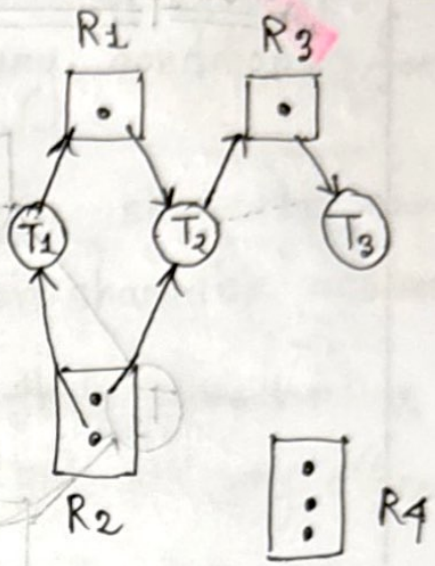
Assign edge | Request edge

** Resource Allocation graph

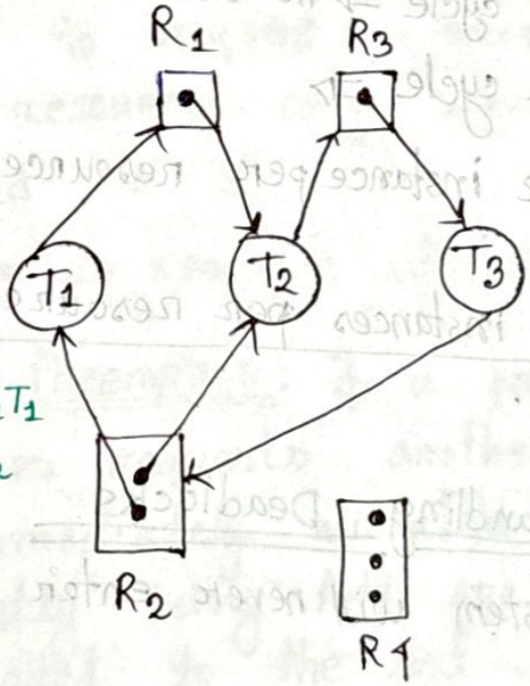
• = instance

Three instances of R_4, two for R_2.
One instance for R_1, R_3.

T_1 holds one instance of R_2 and is waiting for an instance of R_1. T_2 holds one instance of R_1, one instance of R_2 and is waiting for an instance of R_3. T_3 holds one instance of R_3.



*** Resource Allocation graph with a Deadlock



There are 2 cycles:
(i) $T_1 R_1 T_2 R_3 T_3 R_2 T_1$
(ii) $T_2 R_3 T_3 R_2 T_2$

T_1:
Holds: R_2 ($R_2 \rightarrow T_1$)
Requests: R_1 ($T_1 \rightarrow R_1$)

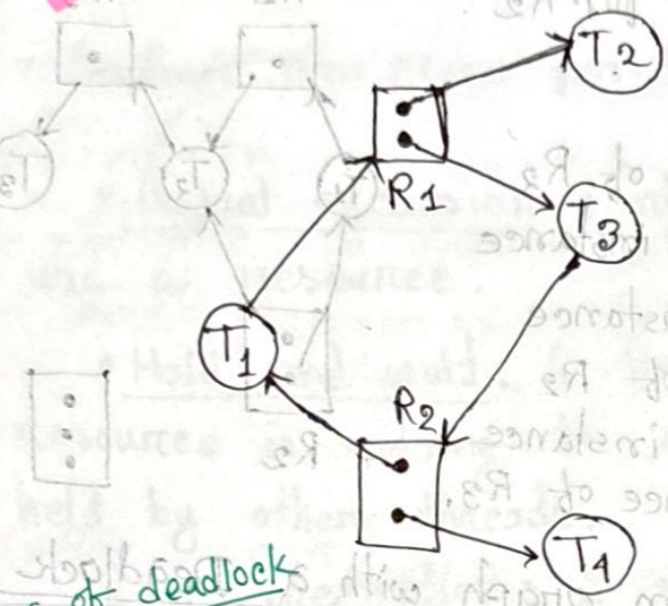
T_2:
Holds: R_1, R_2
Requests: R_3 ($T_2 \rightarrow R_3$)

T_3:
Holds: R_3 ($R_3 \rightarrow T_3$)
Request: R_2 ($T_3 \rightarrow R_2$)

Here, all the threads are holding resources and waiting for other resources. So, there is no progress.

~~to release~~

** Graph with a Cycle But no Deadlock



There is one cycle $T_1 T_3 R_2 T_1$. Here T_2 is not waiting or requested for any resource. T_2 can complete its execution. After T_2 finishes its execution, Then T_2 will release instance of R_1. So, All threads can execute. As a result, there is no deadlock because there is no circular dependency.

Conditions of deadlock in RAG

** If graph contains no cycle ⇒ no deadlock

** If graph contains a cycle ⇒

(1) if only one instance per resource type, then deadlock

(2) if several instances per resource type, possibility of deadlock.

*** Methods for Handling Deadlocks

- Ensure that the system will never enter a deadlock state :
  - Deadlock prevention
  - Deadlock avoidance.
- Allow the system to enter a deadlock state and then recover
- Ignore the problem and pretend that deadlocks never occur in the system.

Deadlock prevention

Invalidate one of the four necessary conditions for deadlock:

- Mutual Exclusion: Not required for shareable resources (read only files); must hold for non-shareable resources.
- Hold and wait: must guarantee that whenever a thread requests a resource, it does not hold any other resources.
  - Require threads to request and be allocated all its resources before it begins execution or allow thread to request ~~a resource, it does not hold any other~~ resources. only when the thread has none allocated to it.
  - Low resource utilization; starvation.
- No Preemption: if a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released. Preempted resources are added to the list of resources of which the thread is waiting. Thread will be restarted only when it can regain old resources, as well as the new ones that it is request
- Circular wait: Impose a total ordering of all resource types, and require that each thread requests resources in an increasing order of enumeration

less / Not important

Circular Wait

Invalidating the circular wait condition is most common. Simply assign each resource (i mutex locks) a unique number.

Resources must be acquired in order:

if:

first_mutex = 1

second_mutex = 5

code for thread_two could not be written as follows.

Deadlock Avoidance

Requires that the system has some additional a priori information available:

- Simplest and most useful model requires that each thread declare maximum number of resources of each type that it may need.
- the deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that there can be a circular wait condition.
- Resource allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

- When a thread requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

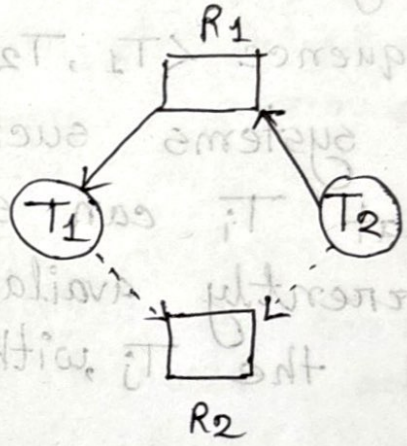
- System is in a safe state if there exists a sequence $\langle T_1, T_2, \ldots, T_n \rangle$ of all the threads in a systems such that for each T_i, the resources that T_i can still request can be satisfied by currently available resources + resources held by all the T_j, with $j < i$

A safe state gurantees that, the system can satisfy all processes resources request in ~~some order~~ ~~eventuall~~ such a way that no deadlock occurs, even if all processes request their maximum resource needs.

Avoidance Algorithm

- Single instance of resource type – Resource allocation graph
- Multiple instances of a resource type – Banker's Algorithm

RAG → Resource Allocation Graph



Banker's Algorithm

- Multiple instances of resources
- Each thread must a priori claim maximum use
- When a thread requests a resource, it may have to wait
- When a thread gets all its resources it must return them in a finite amount of time

- **Available**: How many resources are available?
  - Max
  - Allocation
  - Need = Max - Allocation

Work = Available

If Work ≤ Available

Work = Work + Allocation

Available = Available - Request

Example

3 resource types:
A (10 instances), B (5 instances), C (7 instances)

threads	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
T_0	0	1	0	7	5	3	3	3	2
T_1	2	0	0	3	2	2			
T_2	3	0	2	9	0	2			
T_3	2	1	1	2	2	2			
T_4	0	0	2	4	3	3			

Ans: We know, Need = Max − Allocation

Threads	Need	Condition	if True, w = w + Allocation
T_0	743	743 ≤ 332 False	
T_1	122	122 ≤ 332 True	⇒ Work = 332 + 200 = 532
T_2	600	600 ≤ 532 False	
T_3	011	011 ≤ 532 True	⇒ Work = 532 + 211 = 743
T_4	431	431 ≤ 743 True	⇒ Work = 743 + 002 = 745

We know,
Work = available
Work = 332
∴ Need ≤ Work
Work = Work + Allocati

Again, for T_0, T_2, we find,

$T_0 = 743 \leq 745$ (True) ∴ Work = 745 + 010 = 755

$T_2 = 600 \leq 755$ (True) ∴ Work = 755 + 302 = 1057

Safe Sequence: T_1, T_3, T_4, T_0, T_2

Virtual Memory

Virtual memory is a separation of user logical memory from physical memory, meaning only one part of a program needs to be in RAM for it to run. This allows the program logical address space to be much larger than the physical memory available.

Virtual memory is

Virtual Memory

- Virtual memory is the seperation of user logical memory from the physical memory.
  - Allows more efficient process creation.
  - Allows adress spaces to be shared by several processes.
  - More program running concurrently.
  - Less I/O needed to load or swap processes
  - Logical address can be larger than physical address space.
  - Only part of the program needs to be in memory for execution.

- Virtual memory is a memory managing technique used by modern OS, that ~~creates illusion of larger~~ allows a computer to use disk space as an extension of RAM, enabling larger programs and more processes to run simultaneously.

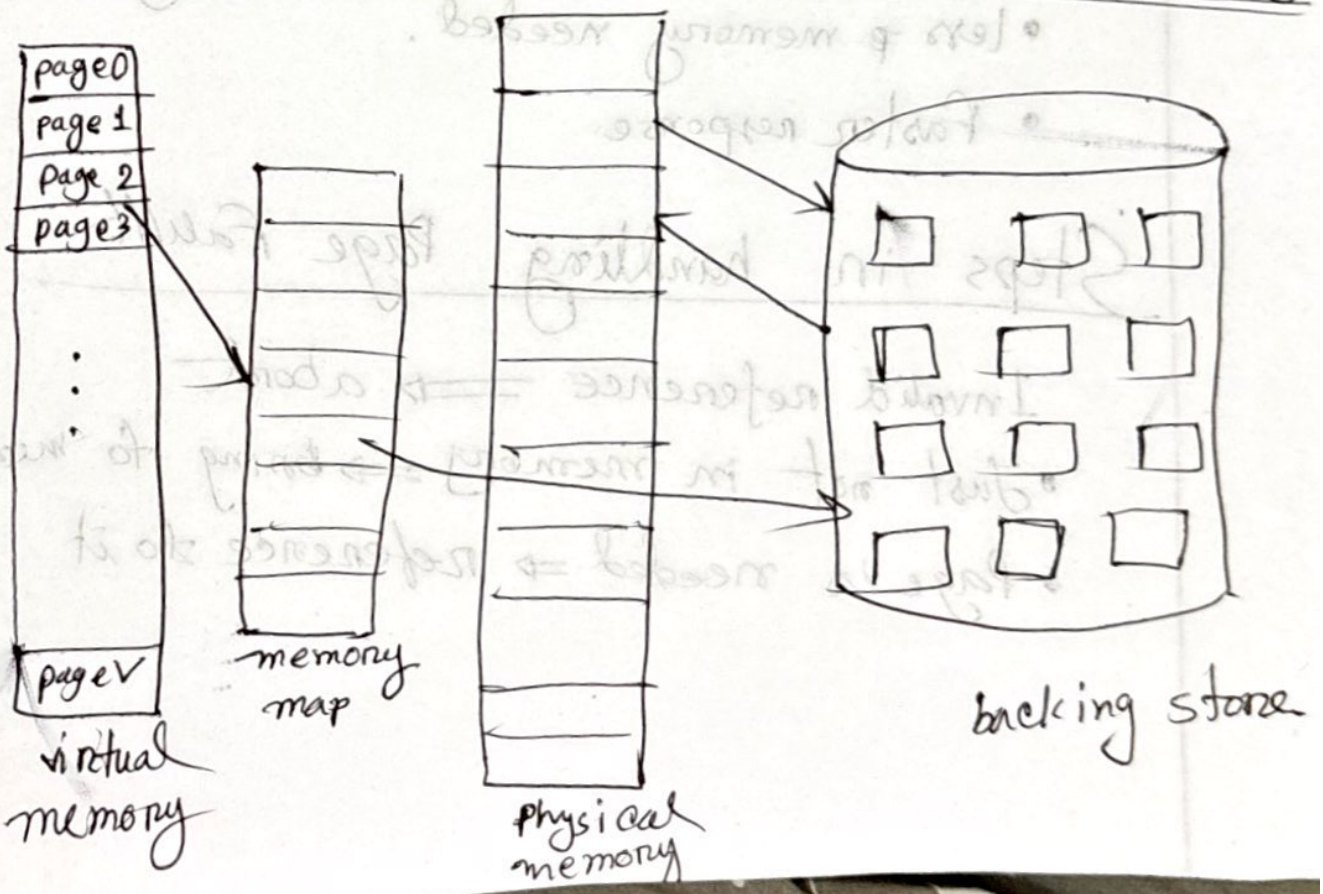
Virtual address space → logical view of how process is stored in memory.

- Usually stated at address 0, contiguous adresses until end of space.
- Physical memory organized in page frames.
- MMU must map logical to pysical.

Virtual memory can be implemented via:

- Demand paging
- Demand segmentation.

Virtual Memory that is larger Than Physical memory



Lazy swapper → never swaps a page into memory unless page will be needed.

- Swapper that deals with pages is a (pager)

Demand Paging

It Could bring entire process into memory at load time, or bring a page into memory only when it is needed.

- less I/O needed, no unnecessary I/O
- less memory needed.
- Faster response

Steps in handling Page Fault

Invalid reference ⟹ abort

- Just not in memory ⟹ bring to memory
- Page is needed ⇒ reference to it