UNIVERSITY INSTITUTE OF ENGINEERING

Bachelor of Engineering (Computer Science & Engineering)

Operating System (CST-328)

Subject Coordinator: Er. Puneet kaur(E6913)

Deadlocks

DISCOVER . LEARN . EMPOWER

Lecture 9

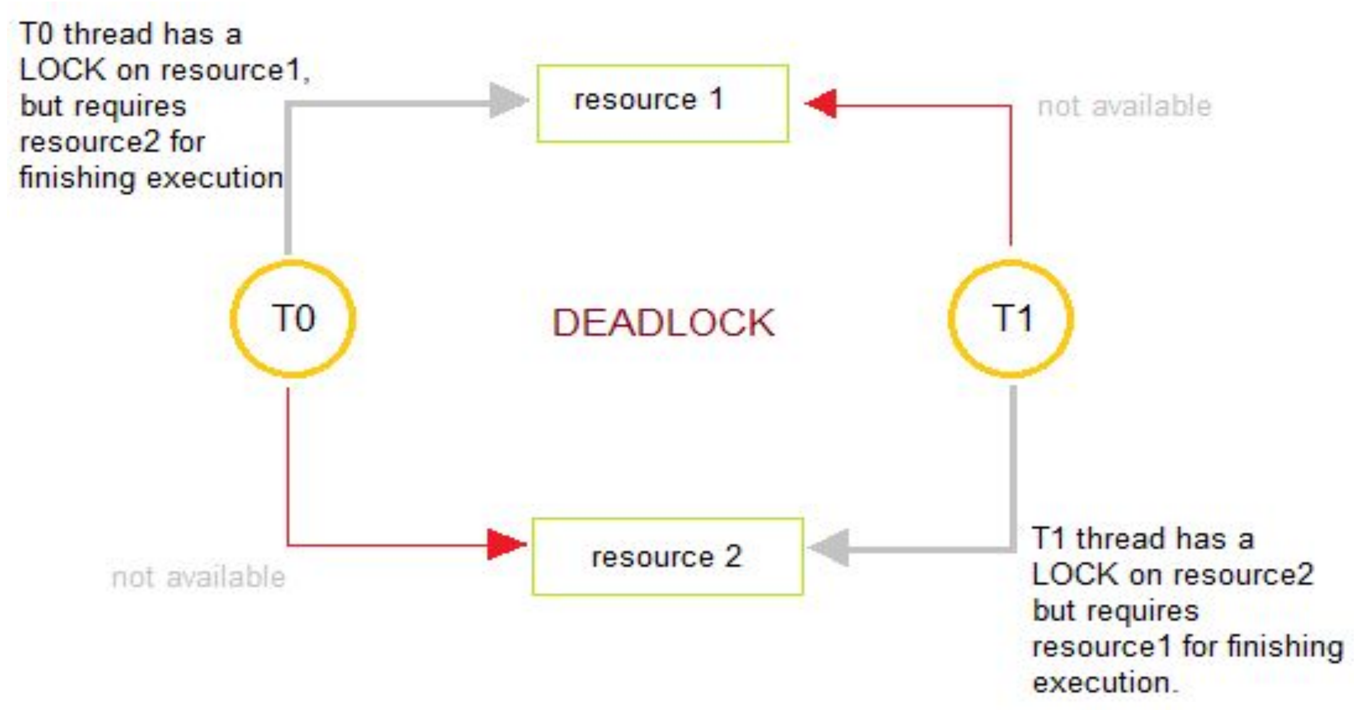
Deadlocks

List-of-content

Deadlocks: Deadlock characterization and conditions for deadlock, deadlock prevention, Deadlock avoidance-safe state, resource allocation graph algorithm, Banker's algorithms-Safety algorithm, Deadlock detection, Recovery from deadlock.

Deadlocks

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

Necessary Conditions for Deadlock

There are four conditions that are necessary to achieve deadlock:

Mutual Exclusion - At least one resource must be held in a non-sharable mode; If any other process requests this resource, then that process must wait for the resource to be released.

Hold and Wait - A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.

No preemption - Once a process is holding a resource (i.e. once its request has been granted), then that resource cannot be taken away from that process until the process voluntarily releases it.

Circular Wait - A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[i] is waiting for P[(i + 1) % (N + 1)]. (Note that this condition implies the hold-and-wait condition, but it is easier to deal with the conditions if the four are considered separately.)

System Model

- Resource types $R_1, R_2, \ldots, R_m$
    - CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
    - request
    - use
    - release

Resource-Allocation Graph

In some cases deadlocks can be understood more clearly through the use of **Resource-Allocation Graphs**, having the following properties:

A set of resource categories, { R1, R2, R3, . . ., RN }, which appear as square nodes on the graph. Dots inside the resource nodes indicate specific instances of the resource. (E.g. two dots might represent two laser printers.)

A set of processes, { P1, P2, P3, . . ., PN }

Resource-Allocation Graph

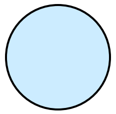
Request Edges - A set of directed arcs from Pi to Rj, indicating that process Pi has requested Rj, and is currently waiting for that resource to become available.

Assignment Edges - A set of directed arcs from Rj to Pi indicating that resource Rj has been allocated to process Pi, and that Pi is currently holding resource Rj.

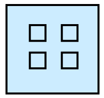
Note that a **request edge** can be converted into an **assignment edge** by reversing the direction of the arc when the request is granted. (However note also that request edges point to the category box, whereas assignment edges emanate from a particular instance dot within the box.)

Resource-Allocation Graph (Cont.)

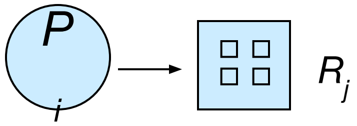
- ❑ Process



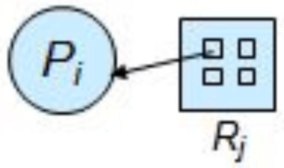
- ❑ Resource Type with 4 instances



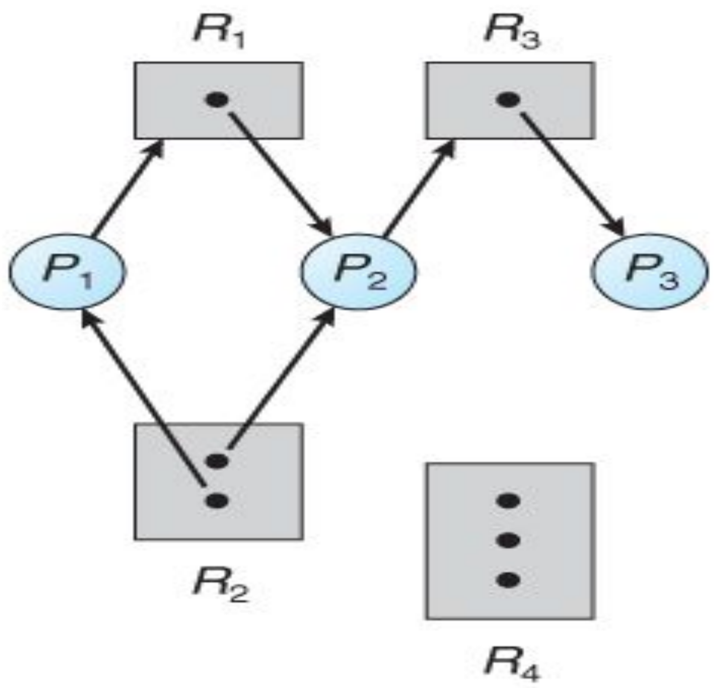
- ❑ P_i requests instance of R_j



- ❑ P_i is holding an instance of R_j

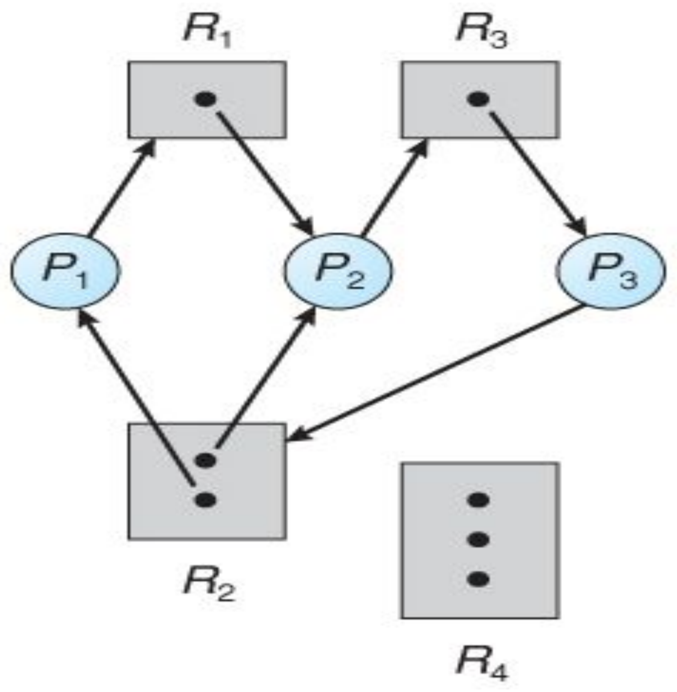


Resource-Allocation Graph (Cont.)



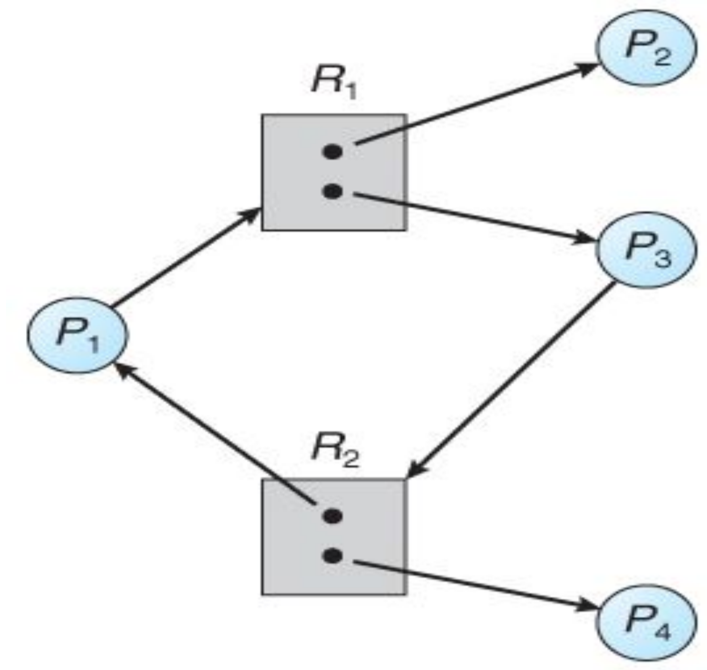
Resource allocation graph

Resource-Allocation Graph (Cont.)



Resource allocation graph with a deadlock

Resource-Allocation Graph (Cont.)



Resource allocation graph with a cycle but no deadlock

Basic Facts

If graph contains no cycles ⇒ no deadlock

If graph contains a cycle ⇒

- if only one instance per resource type, then deadlock
- if several instances per resource type, possibility of deadlock

Important Concept

Consider there are n processes in the system P_1, P_2, P_3, …… , P_n where-

Process P_1 requires x_1 units of resource R

Process P_2 requires x_2 units of resource R

Process P_3 requires x_3 units of resource R and so on.

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

Process P_1 holds $x_1 – 1$ units of resource R

Process P_2 holds $x_2 – 1$ units of resource R

Process P_3 holds $x_3 – 1$ units of resource R and so on.

Now,

If one more unit of resource R is present in the system, then system could be ensured deadlock free.

This is because that unit would be allocated to one of the processes and it would get execute and then release its units.

Important Concept

Maximum Number Of Units That Ensures Deadlock-

Maximum number of units of resource R that ensures deadlock

$= (x_1-1) + (x_2-1) + (x_3-1) + \ldots. + (x_n-1)$

$= (x_1 + x_2 + x_3 + \ldots. + x_n) – n$

$= \sum x_i – n$

= Sum of max needs of all n processes – n

Minimum Number Of Units That Ensures No Deadlock-

Minimum number of units of resource R that ensures no deadlock

= One more than maximum number of units of resource R that ensures deadlock

$= (\sum x_i – n) + 1$

Conclusion

This lecture enables the students to understand what is a deadlock, necessary conditions for deadlock, resource allocation graph other important concepts of deadlocks.

References

https://www.includehelp.com/c-programming-questions/

https://www.studytonight.com/operating-system/

https://computing.llnl.gov/tutorials/

https://www.tutorialspoint.com/operating_system/index.htm#:~:text=An%20operating%20system%20(OS)%20is,software%20in%20a%20computer%20system.

https://www.javatpoint.com/os-tutorial

https://www.guru99.com/operating-system-tutorial.html
https://www.geeksforgeeks.org/operating-systems/