Deadlock - 2

CS3600

Spring 2022

Deadlock Avoidance

- Proactive decision based on
  - Resources currently available
  - The resources currently allocated to each process/thread
  - The future requests and releases of each thread.

A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist.

https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/deadlock-detection

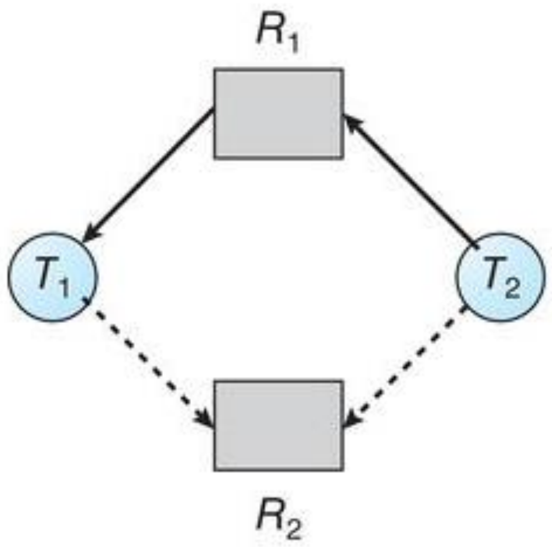
Linux *lockdep tool*

- To detect possible deadlocks, Linux provides lockdep, a tool with rich functionality that can be used to verify locking order in the kernel. lockdep is designed to be enabled on a running kernel as it monitors usage patterns of lock acquisitions and releases against a set of rules for acquiring and releasing locks.

- Its purpose is to test whether software such as a new device driver or kernel module provides a possible source of deadlock.
- https://www.kernel.org/doc/Documentation/locking/lockdep-design.txt.

Resource Claim Graph

- The current allocation of resources to processes and
- All current as well as all potential future requests (dotted line) by processes for new resources.



Banker’s Algorithm

- Multiple instances.
- When a process requests a resource, it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time

Data Structures for the Banker's Algorithm

- Let n = number of processes, and m = number of resources types.
- Max : An n x m matrix for defining the maximum demand of each process.
- Allocation : An n x m matrix for defining the number of allocated resources for each processes
- Available*:* Vector of length m. If k instances of resource type R_j is available
- Need ? $\boxed{\textit{Need }[n,m] = \textit{Max}[n,m] - \textit{Allocation }[n,m]}$

Example

- 5 processes P_0 through P_4 => n= 5
- 3 resource types: *A* (10 instances), *B* (5instances), and *C* (7 instances) => m=3

Allocation 5 x 3

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Max 5 x 3

	A	B	C
P_0	7	5	3
P_1	3	2	2
P_2	9	0	2
P_3	2	2	2
P_4	4	3	3

Available?

A	B	C
3	3	2

Prepare the need matrix

Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*]

Example Cont.

- 5 processes P_0 through P_4
- 3 resource types: A (10 instances), B (5instances), and C (7 instances)

Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*]

Allocation 5 x 3

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Max 5 x 3

	A	B	C
P_0	7	5	3
P_1	3	2	2
P_2	9	0	2
P_3	2	2	2
P_4	4	3	3

Available?

A	B	C
3	3	2

Need?

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Is the current state safe ???

Example Cont.

- 5 processes P_0 through P_4
- 3 resource types: A (10 instances), B (5instances), and C (7 instances)

We will see if the current state is feasible

Allocation 5 x 3

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Need?

A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Available?

A	B	C
10	5	7

P_2

Is Need <= Available ?

Need = (6 0 0) & Available = (7 5 5)

YES

P2 finish =true

Available = Available +Allocation of P2

Order of allocation should be <P1 ,P3 ,P4 ,P0,P2>

Example Cont.

- 5 processes P_0 through P_4
- 3 resource types: A (10 instances), B (5instances), and C (7 instances)

P1 Request for (1 0 2) , Can it be granted???

Allocation 5 x 3

	A	B	C
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Available?

A	B	C
2	3	0

Max 5 x 3

A	B	C
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Need?

A	B	C
7	4	3
0	2	0
6	0	0
0	1	1
4	3	1

Deadlock Prevention

Prevent

- **Mutual exclusion**
  - At least one resource must be non-sharable.
- **Hold and wait**
  - Before execution make sure all resources are available to avoid hold and wait
- **No preemption**
  - If the thread is holding a resource and request a second resource not available now, then all resources will be preempted.
- **Circular wait**

Announcements as of 03/17/2022

- Complete Worksheet 06 and submit in Canvas.
- Programming Project 01– Implementing Banker's algorithm in C [or Any comfortable programming language] using arrays available on 03/19
- Next Class: Memory Management