

Deadlock Solutions

CS 241

Oct. 25, 2013

Review

- Four conditions for deadlock:
 - Hold and Wait
 - No Preemption
 - Circular Wait
 - Mutual Exclusion
- Resource Allocation Graph (RAG) and Wait-For Graph (WFG)
- Introduction to three solutions to deadlock:
 - Deadlock Avoidance
 - Deadlock Prevention
 - Deadlock Detection

Four concurrent processes:

P1: lock (r2) ;
lock (r1) ;

P2: lock (r1) ;
lock (r2) ;

P3: lock (r3) ;
lock (r1) ;

P4: lock (r3) ;
lock (r1) ;

Example Run

```
P1 : lock (r2) ;
P3 : lock (r3) ;
P3 : lock (r1) ;
P2 : lock (r1) ;
P1 : lock (r1) ;
P2 : lock (r2) ;
P4 : lock (r3) ;
P4 : lock (r1) ;
```

Example Run

```
P1 : lock (r2) ;
P2 : lock (r1) ;
P3 : lock (r3) ;
P4 : lock (r3) ;
P1 : lock (r1) ;
P2 : lock (r2) ;
P3 : lock (r1) ;
P4 : lock (r1) ;
```

Deadlock Avoidance

- **Deadlock Avoidance:** Create a system policy that ensures that deadlock can not exist.
 - **How?**
- **Solution:**
 -
 -

Programming w/ Deadlock Avoidance

Four concurrent processes:

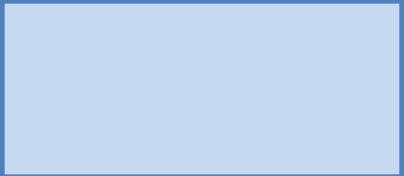
P1: `lock (r2) ;`
`lock (r1) ;`

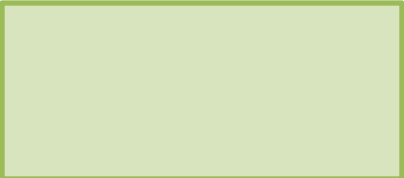
P2: `lock (r1) ;`
`lock (r2) ;`

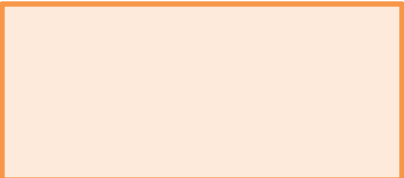
P3: `lock (r3) ;`
`lock (r1) ;`

P4: `lock (r3) ;`
`lock (r1) ;`

Deadlock Free Solution!

P1: 

P2: 

P3: 

P4: 

Deadlock Prevention

- **Deadlock Prevention:** Before assignment a resource, ensure deadlock is not created -- *preventing the deadlock in realtime.*

- **How?**

- **Solutions:**

-

-

Banker's Algorithm (Dijkstra, 1965)

- Three things will always be given:
 - The maximum allocations the system can provide
 - The current allocations already committed
 - The maximum allocations per process
- The Banker's Algorithm takes an **allocation request** and will determine if the requested allocation will leave the system in a **safe** state.
 - **Safe**: A state where there *exists* a path such that all processes will be able to complete

Banker's Algorithm

System Maximum:

A	B	C
10	5	5

A	B	C

Current Allocations

Maximum Required

Unmet Need

	A	B	C
P1:	0	1	0
P2:	2	0	0
P3:	3	0	0
P4:	2	1	1
P5:	0	0	2

	A	B	C
P1:	7	5	3
P2:	3	2	2
P3:	9	0	2
P4:	2	2	2
P5:	4	3	3

	A	B	C
P1:			
P2:			
P3:			
P4:			
P5:			

P2 Requests: (A: 1, B: 0, C: 2)

Banker's Algorithm

System Maximum:

A	B	C
10	5	5

A	B	C

Current Allocations

Maximum Required

Unmet Need

	A	B	C
P1:	0	1	0
P2:	2	0	0
P3:	3	0	0
P4:	2	1	1
P5:	0	0	2

	A	B	C
P1:	7	5	3
P2:	3	2	2
P3:	9	0	2
P4:	2	2	2
P5:	4	3	3

	A	B	C
P1:			
P2:			
P3:			
P4:			
P5:			

1. P2 Requests: (A: 1, B: 0, C: 2)
2. P1 Requests: (A: 0, B: 2, C: 0)

Banker's Algorithm

System Maximum:

A	B	C
8	4+y	10

A	B	C

Current Allocations

Maximum Required

Unmet Need

	A	B	C
P1:	2	0	2
P2:	0	0	2
P3:	2	2	2

	A	B	C
P1:	4	2	4
P2:	6	4	8
P3:	8	4	2

	A	B	C
P1:			
P2:			
P3:			

P2 Requests: (A: 1, B: 0, C: 1)

Q: What is the minimum value of y such that the request can be granted?

Deadlock Detection

- **Deadlock Detection:** Allow deadlocks to occur and detect them in the system after they have occurred.
 - **How?** Run any of the previous solutions at a regular intervals.
 - Wait-for-Graph
 - Banker's Algorithm

Deadlock Solutions

- **Deadlock Avoidance:**
- **Deadlock Prevention:**
- **Deadlock Detection:**