CS302
Operating System
Lab 7
Synchronization

Xinxun Zeng

Deal with deadlock

- We can use a protocol to **prevent** or **avoid** deadlocks, ensuring that the system will never enter a deadlocked state.

- We can allow the system to enter a deadlocked state, **detect it, and recover**.

- We can **ignore** the problem altogether and pretend that deadlocks never occur in the system.

Other alternative way

1. Setting grab order
   - Odd number philosopher first take left and then right
   - Even number philosopher first take right and then left
2. Allow at most 4 philosopher take the left chopsticks at the same time
   - Add a semaphore to monitor this behavior

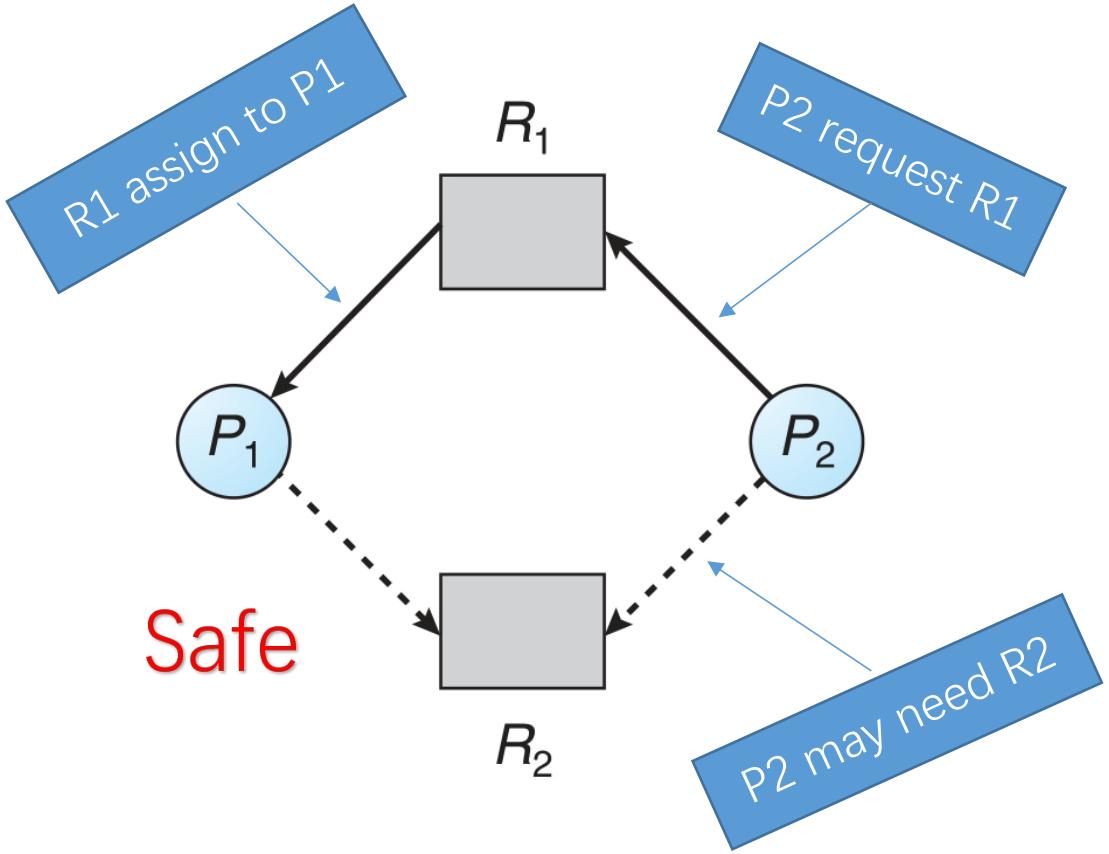
Deadlock Prevention

- Break the one of the requirement of deadlock
  - Mutual exclusion
    - Only one thread at a time can use a resource.
  - Hold and wait
    - Thread holding at least one resource is waiting to acquire additional resources held by other threads
  - No preemption
    - Resources are released only voluntarily by the thread holding the resource, after thread is finished with it
  - Circular wait
    - There exists a set $\{T_1, \ldots, T_n\}$ of waiting threads
      - T_1 is waiting for a resource that is held by T_2
      - T_2 is waiting for a resource that is held by T_3

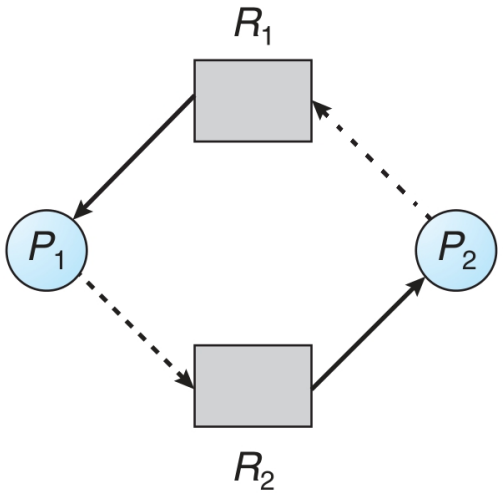
Deadlock Avoidance

- Resource-Allocation-Graph Algorithm
- Banker's Algorithm

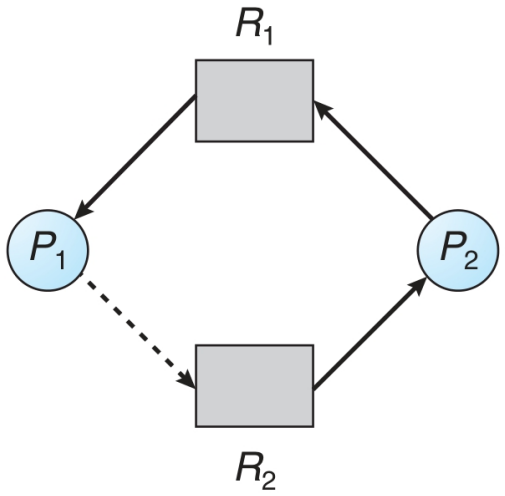
Resource-Allocation-Graph Algorithm



Safe



Unsafe



Deadlock

Time complexity $O(n^2)$

Banker’s Algorithm

- Safety Checking
- Resource Requesting

Safe State

- A state is safe if the system can allocate resources to each process (up to its maximum) in a safe sequence.
- Assume we have 12 resources

Process	Maximum Need	Current Allocated
P1	10	0
P2	4	0
P3	9	0
Remain		12

Safe

Process	Maximum Need	Current Allocated
P1	10	5
P2	4	2
P3	9	2
Remain		3

Safe

Process	Maximum Need	Current Allocated
P1	10	5
P2	4	2
P3	9	4
Remain		2

Unsafe

Safe State (more example)

Process	Maximum Need	Current Allocated
P1	10	3
P2	4	0
P3	9	5
Remain		4

Safe

Process	Maximum Need	Current Allocated
P1	10	6
P2	4	4
P3	9	2
Remain		0

Safe

Process	Maximum Need	Current Allocated
P1	10	4
P2	4	4
P3	9	4
Remain		0

Unsafe

Safety Checking

	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P1	7	5	3	0	1	0	7	4	3
P2	3	2	2	2	0	0	1	2	2
P3	9	0	2	3	0	2	6	0	0
P4	2	2	2	2	1	1	0	1	1
P5	4	3	3	0	0	2	4	3	1
Remain				3	3	2			

Safe

P2, P4, P1, P3, P5

Time complexity $O(mn^2)$

Resource Requesting

	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P1	7	5	3	0	1	0	7	4	3
P2	3	2	2	2	0	0	1	2	2
P3	9	0	2	3	0	2	6	0	0
P4	2	2	2	2	1	1	0	1	1
P5	4	3	3	0	0	2	4	3	1
Remain				3	3	2			

New request: P2 (1,0,2) <Remain? <Max(P2)?

Resource Requesting

	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P1	7	5	3	0	1	0	7	4	3
P2	3	2	2	3	0	2	0	2	0
P3	9	0	2	3	0	2	6	0	0
P4	2	2	2	2	1	1	0	1	1
P5	4	3	3	0	0	2	4	3	1
Remain				2	3	0			

New request: P2 (1,0,2) <Remain? <Max(P2)?

Try assign resource for P2 Still safe? Safe!

Resource Requesting

	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P1	7	5	3	0	1	0	7	4	3
P2	3	2	2	2	0	0	1	2	2
P3	9	0	2	3	0	2	6	0	0
P4	2	2	2	2	1	1	0	1	1
P5	4	3	3	0	0	2	4	3	1
Remain				3	3	2			

New request: P2 (2,0,2) <Max(P2)? ×

New request: P5 (4,0,0) <Remain? ×

New request: P1 (3,3,2) Still safe? ×

Dining Philosophers with Banker's Algorithm

	Max			Allocation			Need		
	C1	C2	C3	C1	C2	C3	C1	C2	C3
P1	1	1	0	0	0	0	1	1	0
P2	0	1	1	0	0	0	0	1	1
P3	1	0	1	0	0	0	1	0	1
Remain				1	1	1			

Assignment: Banker's Algorithm

- You are asked to implement a Banker's Algorithm, your code will be judged by standard test case.
- language: C++, you are free to use all STL
- Input:
  - First line is an integer r, which is the number of resource types.
  - The second line will be r integers, which are the maximum quantity of each resource.
  - Then will be following a list of commands. The commands are in three form:
    1. New process registering, such as "1678 new 6 5 0 7", means process 1678 is a new process, whose maximum need of each resource is 6 5 0 7 (assume r is 4)
    2. Resource requesting, such as "233 request 0 4 5 3", means process 233 is an old process, it request more resource, the need of each resource is 0 4 5 3.
    3. Process termination, such as "233 terminate", means process 233 terminate and return all resources it holding.
- Output:
  - For each command, output "OK" or "NOT OK" to determine if the command can execute. If OK, execute the command.

Assignment Sample

- input:

```
3
4 2 3
233 new 1 2 3
888 new 4 3 3
777 new 4 2 3
233 request 1 2 0
777 request 0 0 4
777 request 0 0 3
233 terminate
```

- output:

```
OK
NOT OK
OK
OK
NOT OK
NOT OK
OK
```