

# COSC 450 Mid2 Studying cont.

## Deadlock Stuff (Detection & Recovery)

- Four strategies for dealing w/ deadlock:

- Just ignore
- Detection & Recover
- Dynamic Avoidance by Careful Allocation
- Prevention - negate one of the four deadlock conditions

- OS keeps matrices for deadlock detection algorithm:

→ Existing Resource Matrix (one dimension, E)

↳ Number of resources per each type

→ Available Resource Matrix (one dimension, A)

↳ Number of resources available at any moment

→ Current Allocation Matrix (two dimension, C)

↳ Row: in matrix presents resources currently held by process P<sub>i</sub>

→ Request Matrix (two dimension, R)

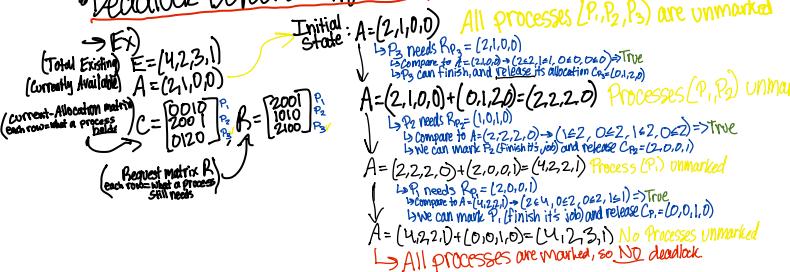
↳ row: in R presents how many more resources are needed for process P<sub>i</sub> to finish its job.

- The deadlock detection algorithm is based on comparing vectors.

↳ If we define the relation A ≤ B between two one-dimensional vectors (A & B), then each element of A is less than or equal to the corresponding element of B.

- Deadlock Detection Algorithm

- Actually really easy, just understand



## Recovery Methods from Deadlock

- Recovery through preemption

↳ Take a resource from some other process

↳ Depends on nature of the resource

- Recover through rollback

↳ checkpoint a process periodically

↳ restore the process if it is found deadlocked

- Recovery through killing process

↳ crudest but simplest way to break a deadlock

↳ kill one of the processes in the deadlock cycle

↳ the other process gets its resources

↳ the killed process starts from the beginning

## For detection + Bankers

↳ D (often called Need)

$$R_{ij} = M_{ij} - C_{ij}$$

↳ Maximum

$$\rightarrow A = E - EC$$

Add all columns of C

Get A by subtracting

E (using) by the sum of all columns of C

Currently allocated

$$\rightarrow A = E - EC$$

$$\rightarrow R = M - C$$

↳ Maximum Demand Matrix

## Deadlock Conditions

- Mutual Exclusion
- Hold and Wait
- No preemption
- Circular Wait

↳ Only occurs if all are true simultaneously

MEMORIZE

↳ since some resources are assigned to process but is not available

↳ If  $A \leq R$  then no deadlock

↳ If  $A > R$  then deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

↳ If  $A < R$  then no deadlock

↳ If  $A = R$  then deadlock

## Given Scenario, is it deadlock?

(probably like 40% chance of being asked, brief understanding of concepts)

Ex) Scenario:  $P_1, P_2, P_3$  competing for two non-preemptive resource  $R_1$  and  $R_2$ , each w/ exactly one instance.

At time t, the system states:

$P_1$  holds  $R_1$  and is waiting for  $R_2$

$P_2$  holds  $R_2$  and is waiting for  $R_1$

$P_3$  holds no resource and is waiting for  $R_2$

Solution:

1. Draw the Resource-Allocation Graph

Processes:  $P_1, P_2, P_3$  Resources:  $R_1, R_2$

$P_1 \rightarrow P_1$  (P<sub>1</sub> holds R<sub>1</sub>)

$P_2 \rightarrow P_2$  (P<sub>2</sub> holds R<sub>2</sub>)

$P_1 \rightarrow P_2$  (P<sub>1</sub> waiting for R<sub>2</sub>)

$P_2 \rightarrow P_1$  (P<sub>2</sub> is holding for P<sub>1</sub>)

$P_3 \rightarrow P_3$  (P<sub>3</sub> is also waiting for R<sub>2</sub>)

Since the resulting Resource-Allocation graph has a cycle, this indicates deadlock.

2. Verifying the four necessary conditions

Deadlock only occurs if all four conditions are met.

① Mutual Exclusion - Both  $P_1$  &  $P_2$  are non-preemptive, not being able to be shared or taken away. If a request is made for a resource, it must wait, creating potential for blocking.

↳ Met

② Hold and Wait -  $P_1, P_2$  holds  $R_1, R_2$  will attempt to request the other one. Since each process is stuck holding & waiting, neither process can advance.

↳ Met

③ No Preemption - Neither  $P_1$  nor  $P_2$  can be taken/stolen from  $P_1$  or  $P_2$ , leaving no mechanisms to potentially break infinite holds.

↳ Met

④ Circular Wait - The resource allocation graph directly shows a cycle, where no process can ever acquire the next needed resource, instead waiting on others infinitely.

↳ Met

Therefore, since all four deadlock conditions are met, this scenario is deadlock.