COSC450 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 matrixes 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 (since some resources are assigned to process but is not available)
  - → Current Allocation Matrix (two dimension, C)
    - ↳ row$_i$ in matrix presents resources currently held by process P_i
  - → Request Matrix (two dimension, R)
    - ↳ row$_i$ in R presents how many more resources are needed for process P_i to finish it's job.
- The deadlock detection algorithm is based on comparing vectors.
  - ↳ If we define the relation $A \leq B$ between two one-dimensional vectors (A & B), then each element of A is less than or equal to the corresponding element of B.

Deadlock Conditions

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

↳ only occurs if all are true simultaneously

Memorize

- Deadlock Detection Algorithm - Actually really easy, just understand

→ Ex)

(Total Existing) E=(4,2,3,1)

(Currently Available) A=(2,1,0,0)

(current Allocation matrix, each row=what a process holds) $C=\begin{bmatrix}0&0&1&0\\2&0&0&0\\0&1&2&0\end{bmatrix}$ P_1 P_2 P_3

(Request matrix R, each row=what a process still needs) $R=\begin{bmatrix}2&0&0&1\\1&0&1&0\\2&1&0&0\end{bmatrix}$ P_1 P_2 P_3

Initial State: A=(2,1,0,0) All processes (P_1, P_2, P_3) are unmarked

↳ P_3 needs R_{P_3} = (2,1,0,0)
↳ Compare to A=(2,1,0,0) → (2≥2, 1≥1, 0≥0, 0≥0) => True
↳ P_3 can finish, and release its allocation C_{P_3}=(0,1,2,0)

A=(2,1,0,0)+(0,1,2,0)=(2,2,2,0) Processes (P_1, P_2) unmarked

↳ P_2 needs R_{P_2} = (1,0,1,0)
↳ Compare to A=(2,2,2,0) → (1≤2, 0≤2, 1≤2, 0≤0) => True
↳ We can mark P_2 (finish it's job) and release C_{P_2}=(2,0,0,1)

A=(2,2,2,0)+(2,0,0,1)=(4,2,2,1) Process (P_1) unmarked

↳ R_1 needs R_{P_1} = (2,0,0,1)
↳ Compare to A=(4,2,2,1) → (2≤4, 0≤2, 0≤2, 1≤1) => True
↳ We can mark P_1 (finish it's job) and release C_{P_1}=(0,0,1,0)

A=(4,2,2,1)+(0,0,1,0)=(4,2,3,1) No Processes unmarked

↳ All processes are marked, so NO deadlock.

Ex)

Existing E=(4,2,3,1)

Available A=(2,1,0,0)

Current Allocation $C=\begin{bmatrix}0&0&1&0\\2&0&0&0\\0&1&2&0\end{bmatrix}$ P_1 P_2 P_3

Request $R=\begin{bmatrix}2&0&0&1\\1&0&1&1\\2&1&0&0\end{bmatrix}$ P_1 P_2 P_3

unmarked = P_1, P_2, P_3

Initial State: A=(2,1,0,0)
↳ P_3 is only process who's Requests ≤ Available
↳ So P_3 can run & release C_{P_3}=(0120)

A=(2,1,0,0)+(0,1,2,0)=(2,2,2,0) (Available)
↳ However, now NO process requests are ≤ A
↳ No rows in R
↳ Therefore, Deadlock

P_3 ≤ A — If any deadlock

- Recovery methods from Deadlock
  - → Recovery through preemption
    - ↳ Take a resource from some other process
    - ↳ Depends on nature of the resource
  - → Recovery through rollback
    - ↳ Checkpoint a process periodically
    - ↳ Use this saved state
    - ↳ restart 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 it's resources
    - ↳ The killed process starts from the beginning
- For detection + Bankers
  - ↳ R (often called Need)
  - $R_{i,j} = M_{ij} - C_{i,j}$ (M = Maximum)
  - ↳ $A = E - \sum C$ (Add all columns of C; Get A by subtracting E (existing) by the sum of all columns of C (currently allocated))
  - → $A = E - \sum C$
  - → $R = M - C$ (M = Maximum Demand Matrix)

Given Scenario, is it deadlock?

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

Ex) Scenario: (P_1, P_2, P_3)

Consider three processes competing for two non-preemptive resources R_1 and R_2, each w/ exactly one instance.

At time t_0, 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 resources 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

Remember: P=set of process, R=set of resources; P→R means P request resource R and is currently waiting; R→P means resource R is currently held by process P

Assignment edges:
- $R_1 \rightarrow P_1$ (P_1 holds R_1)
- $R_2 \rightarrow P_2$ (P_2 holds R_2)

Request edges:
- $P_1 \rightarrow R_2$ (P_1 waiting for R_2)
- $P_2 \rightarrow R_1$ (P_2 is waiting for R_1)
- $P_3 \rightarrow R_2$ (P_3 is also waiting for R_2)

↳ cycle in graph

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 R_1 & R_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 attempting to request the other one. Since each process is stuck holding & waiting, neither process can advance
↳ Met

③ No Preemption - Neither R_1 nor R_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.