Ch 3

– What is a deadlock?
– Conditions
  • Hold and Wait
  • Mutual Exclusion
  • Non Preemption
  • Circular Wait
– Deadlock Models
  • Single Unit Request
  • AND Request
  • OR Request
  • AND-OR Request
  • P-out of-Q Request
• Resource Models
  – Reusable – fixed number of units which can neither be created nor destroyed. Unit available after release from process.
  – Consumable – is used up by a process and no longer available. Are “produced” as well.

• Resource Access
  – Exclusive or Shared

• Miscellany: Wait For Graphs (WFG)
  – Cycles ? Knots ?
General Resource Graph

- Bipartite Directed Graph
  - Vertices are:
    - $P = \text{set of processes } P_1 \cdots P_n$
    - $R = \text{set of resources } R_1 \cdots R_n$
      - Can be subdivided into disjoint sets of consumable and reusable
      - For every reusable resource $R_i$, $t_i$ denotes total number of $R_i$
  - Edges are:
    - Request -- directed from $P$ to $R$
    - Assignment -- directed from reusable $R$ to $P$
    - Producer -- directed from consumable $R$ to $P$
  - Available Unites vector
    - $(r_1 \cdots r_n)$ of nonnegative integers denotes instances of resource available in a given state.
- For every reusable resource
  - No. of assignment edges $\leq ti$
  - $ri = ti - $ No. of assignment edges
  - At any instant, a process cannot request more than the total no. of resources $\#(P_j, R_i) + \#(R_i, P_j) \leq ti$.

- For every consumable resource, $ri \geq 0$.

- A process can request resources, acquire a resource, and release it. These will lead to changes in the graph.
  - Request will add request edges. Assignment will convert request edges to assignment edges for reusables, delete them for consumables, and decrease $r$.
  - Release occurs when the process does not need $R_j$ anymore. $r_j$ is incremented (differently for reusables and consumables).
Conditions for Deadlock

- Process is blocked if the number of its request edges for some \( R_j \) is greater than \( r_j \), the number available.
- This will lead to a deadlock iff it can’t become unblocked eventually.
  - Can you “reduce” the GRG to unblock the process?
- An unblocked process \( P_i \) can reduce the GRG as follows
  - For each reusable resource \( R_j \), delete assignment (and request) edges from \( P_i \), and increment \( r_j \) by the number of assignment edges deleted
  - For each consumable resource, decrement \( r_j \) by the number of request edges. If \( P_i \) is a producer of \( R_j \), set \( r_j \) to “infinity”. Delete request edges.
Sufficiency Conditions

- A GRG is *completely reducible* if some sequence of reductions will delete all edges.
- Theorem: A process is not deadlocked iff some sequence of reductions will leave it unblocked.
- Corollary: A system state is deadlock free if the GRG is completely reducible.
  - Reverse is not true – non reducibility does not imply that a state is deadlocked.
- Detecting deadlocks $\Rightarrow$ investigating $n!$ reduction sequences.
− A state is expedient if all processes having outstanding requests are blocked
− \( X \rightarrow Y \) implies reachability.
− Sink, Cycle, Knot
− A Sink can’t be in a knot
− An “active process” is a sink – reducing is basically removing sink nodes from the graph.
− Theorem: In a GRG
  • A Cycle is a necessary condition for deadlock
  • If the graph is expedient then a knot is a sufficient condition for deadlock.
− Corollary: If in an expedient resource graph, \( P_i \) is not a sink nor does it have a path leading to a sink then the process is deadlocked.
- For Single Unit Requests
  - An expedient GRG with SU Requests represents a deadlock if it contains a knot.

- Systems with Consumable Resources only
  - Claim limited graph represents a worst case condition – no resources are available
  - If this claim limited graph is reducible, then the system is deadlock free. This requires that there be a producer which is not a consumer.

- Systems with Reusable Resources only
  - All reduction sequences give the same outcome.
  - A state is not deadlock state iff it is completely reducible.

- Systems with Single Unit Resources
  - Cycle is necessary and sufficient condition.
• So far, we have looked at Deadlock Detection
• Deadlock Prevention
  – Eliminate one of the 4 necessary conditions.
    • One shot allocation, preemption, resource ordering
• Deadlock Avoidance.
  – When a process requests resources, check to see if the allocation would lead to a safe state. Don’t allocate otherwise. Requires advance knowledge of claims.
    • Be familiar with Banker’s algorithm.