Deadlock

Example

Three Deadlocked Processes

A Model

State Transitions

Properties of States
Properties of States (cont)

- If $p_i$ is blocked in $S_j$, and will also be blocked in every $S_k$ reachable from $S_j$, then $p_i$ is deadlocked.
- $S_j$ is called a **deadlock state**.

A Simple Process-Resource Model Instance

Example

- One process, two units of one resource
- Can request one unit at a time

Extension of Example

Dining Philosophers Revisited

```c
philosopher(int i) {
    while (TRUE) {
        // Think
        // Eat
        P(fork[i]); /* Pick up left fork */
        P(fork[(i+1) mod 5]); /* Pick up right fork */
        eat();
        V(fork[(i+1) mod 5]);
        V(fork[i]);
    }
}

philosopher4() {
    while (TRUE) {
        // Think
        // Eat
        P(fork[0]); /* Pick up right fork */
        P(fork[4]); /* Pick up left fork */
        eat();
        V(fork[4]);
        V(fork[0]);
    }
}
semaphore fork[5];
fork(philosopher, 1, 0);
fork(philosopher, 1, 1);
fork(philosopher, 1, 2);
fork(philosopher, 1, 3);
fork(philosopher4, 0);
```
Addressing Deadlock

- Prevention: Design the system so that deadlock is impossible
- Avoidance: Construct a model of system states, then choose a strategy that will not allow the system to go to a deadlock state
- Detection & Recovery: Check for deadlock (periodically or sporadically), then recover
- Manual intervention: Have the operator reboot the machine if it seems too slow

Prevention

- **Necessary conditions for deadlock**
  - Mutual exclusion
  - Hold and wait
  - Circular waiting
  - No preemption
- Ensure that at least one of the necessary conditions is false at all times
  - Mutual exclusion must hold at all times

Hold and Wait

- Need to be sure a process does not hold one resource while requesting another
- **Approach 1**: Force a process to request all resources it needs at one time
- **Approach 2**: If a process needs to acquire a new resource, it must first release all resources it holds, then reacquire all it needs
- What does this say about state transition diagrams?

Circular Wait (cont)

- Have a situation in which there are K processes holding units of K resources

Circular Wait (cont)

- There is a cycle in the graph of processes and resources
- Choose a resource request strategy by which no cycle will be introduced
- **Total order** on all resources, then can only ask for \( R_i \) if \( R_i < R_j \) for all \( R_i \) the process is currently holding
- This is how we noticed the easy solution for the dining philosophers
Allowing Preemption

- Allow a process to time-out on a blocked request -- withdrawing the request if it fails

Avoidance

- Define a model of system states, then choose a strategy that will guarantee that the system will not go to a deadlock state
- Requires extra information, e.g., the maximum claim for each process
- Allows resource manager to see the worst case that could happen, then to allow transitions based on that knowledge

Safe vs Unsafe States

- **Safe state**: one in which the system can assure that any sequence of subsequent transitions leads back to the initial state
  - Even if all exercise their maximum claim, there is an allocation strategy by which all claims can be met
- **Unsafe state**: one in which the system cannot guarantee that the system will transition back to the initial state
  - Unsafe state *can* lead to a deadlock state if too many processes exercise their maximum claim at once

More on Safe & Unsafe States

- **Banker’s Algorithm**
  - Let maxc[i, j] be the maximum claim for R_j by p_i
  - Let alloc[i, j] be the number of units of R_j held by p_i
  - Can always compute
    - avail[j] = c_j - \sum_{i,j} alloc[i,j]
    - Then number of available units of R_j
  - Should be able to determine if the state is safe or not using this info
Banker’s Algorithm

- Copy the alloc[i,j] table to alloc’[i,j]
- Given C, maxc and alloc’, compute avail vector (noting that the avail only depends upon C and alloc)
- Find p: maxc[i,j] - alloc'[i,j] ≤ avail[j] for 0 ≤ j < m and 0 ≤ i < n.
  - If no such p exists, the state is unsafe
- Set alloc'[i,j] to 0; deallocate all resources held by p; go to Step 2

Detection & Recovery

- Check for deadlock (periodically or sporadically), then recover
- Can be far more aggressive with allocation
- No maximum claim, no safe/unsafe states
- Differentiate between
  - Serially reusable resources: A unit must be allocated before being released
  - Consumable resources: Never release acquired resources; resource count is number currently available

Example

C = <8, 5, 9, 7>

• Compute total allocated
• Determine available units
  avail = <8-7, 5-3, 9-7, 7-5> = <1, 2, 2, 2>
• Can anyone’s maxc be met?
  maxc[2,0]-alloc'[2,0] = 0-1 = -1
  maxc[2,1]-alloc'[2,1] = 0-2 = -2
  maxc[2,2]-alloc'[2,2] = 0-0 = 0
  maxc[2,3]-alloc'[2,3] = 0-3 = -3
  • P2 can exercise max claim

Reusable Resource Graphs (RRGs)

• Micro model to describe a single state
• Nodes = {p0, p1, …, pn} ∪ {R0, R1, …, Rm}
• Edges connect pi to Rj or Rj to pi
  - (pi, Rj) is a request edge for one unit of Rj
  - (Rj, pi) is an assignment edge of one unit of Rj
• For each Rj there is a count, cj, of units Rj
• Number of units of Rj allocated to pi plus the number requested by pi cannot exceed cj
State Transitions due to Request

• In $S_j$, $p_i$ is allowed to request $q \leq c_h$ units of $R_h$, provided $p_i$ has no outstanding requests.

• $S_j \rightarrow S_k$, where the RRG for $S_k$ is derived from $S_j$ by adding $q$ request edges from $p_i$ to $R_h$. 

State Transition for Acquire

• In $S_j$, $p_i$ is allowed to acquire units of $R_h$ iff there is $(p_i, R_h)$ in the graph, and all can be satisfied.

• $S_j \rightarrow S_k$, where the RRG for $S_k$ is derived from $S_j$ by changing each request edge to an assignment edge.

State Transition for Release

• In $S_j$, $p_i$ is allowed to release units of $R_h$, iff there is $(R_h, p_i)$ in the graph, and there is no request edge from $p_i$.

• $S_j \rightarrow S_k$, where the RRG for $S_k$ is derived from $S_j$ by deleting all assignment edges.

Example

- P holds one unit of $R$
- P requests one unit of $R$

A Deadlock State

Not a Deadlock State

No Cycle in the Graph
Example

Graph Reduction
• Deadlock state if there is no sequence of transitions unblocking every process
• A RRG represents a state; can analyze the RRG to determine if there is a sequence
• A graph reduction represents the (optimal) action of an unblocked process. Can reduce by \( p_i \) if
  – \( p_i \) is not blocked
  – \( p_i \) has no request edges, and there are \((R_j, p_i)\) in the RRG
Graph Reduction (cont)

- Transforms RRG to another RRG with all assignment edges into $p_i$ removed
- Represents $p_i$ releasing the resources it holds

\[ \text{Reducing by } p_i \]

Graph Reduction (cont)

- A RRG is completely reducible if there are sequences of reductions that lead to a RRG with no edges
- A state is a deadlock state if and only if the RRG is not completely reducible.

Example RRG

\[ \text{Example RRG} \]

Recovery

- No magic here
  - Choose a blocked resource
  - Preempt it (releasing its resources)
  - Run the detection algorithm
  - Iterate if until the state is not a deadlock state