Deadlocks

- A situation where nobody can make progress
- Examples?

Deadlock Example

Thread A:
lock(mut1);
lock(mut2);

Thread B:
lock(mut2);
lock(mut1);

- What is the problem here?
Dining Philosophers Problem

- N philosophers sitting around a table
- Bowl of noodles in front of each
- One chopstick between each pair
- Each philosopher can think or eat
  - Eating requires both chopsticks
- Only one philosopher can pick a chopstick at a time
- How do we synchronize the philosophers?

Deadlock Definition

- Every thread/process in a set is waiting for an event by another thread/process in the set
- Events: Resource acquisition and release
  - Physical resources: CPU, memory, I/O devices
  - Logical resources: Files, semaphores, locks

Deadlock: Necessary Conditions

- Mutual Exclusion
  - Exclusive access to a resource
- Hold and wait
  - A thread must be holding a resource and waiting for another
- No preemption
  - Only the holding thread can release a resource
- Circular wait
  - A chain of threads must be waiting for each other in a circular manner (one waiting for the next)

Resource Allocation Graph

- Vertices: Threads and resources
- Directed edges:
  - Request edge Ti->Rj: Thread Ti waiting on resource Rj
  - Assignment edge Rj->Ti: Resource Rj allocated to Ti
Resource Allocation Graph:

- **Deadlock**
  - Cycle => deadlock may be there
  - Necessary and sufficient condition if single-instance resources
  - Necessary condition if multi-instance resources

Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery
- Ostrich Algorithm

Deadlock Prevention
- Prevent the system from going into deadlock
- Avoid one of the deadlock conditions
  - Which ones can’t we avoid generally?
- Different techniques designed to avoid different conditions
  - Tradeoff in terms of efficiency
Backoff

```c
while (1) {
    lock(mut1);
    if (trylock(mut2)==-1)
        unlock(mut1);
    else
        break;
}
```

- Check if request will result in blocking
- Release resources if blocking would have occurred
- Ensures that thread either holds all resources or none
- Which condition does it avoid?

Total Ordering

<table>
<thead>
<tr>
<th>Thread A:</th>
<th>Thread B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(mut1); lock(mut2);</td>
<td>lock(mut1); lock(mut2);</td>
</tr>
</tbody>
</table>

- Global ordering of resources
- Acquire the resources in a fixed order
  - E.g.: Each thread acquires mut1 before mut2
- Which condition does it avoid?

Deadlock Avoidance

- Deadlocks possible but system actively avoids them
- Process should give advance info about its resource requests
  - Maximum no. of each resource it needs
- System uses this info to determine possible future states
  - Delays requests that would cause deadlock

Resource Allocation State

- Safe state:
  - System can allocate resources to threads in some order while avoiding deadlock
- Unsafe state:
  - System could go into deadlock at some point
- Deadlock avoidance:
  - Avoid going into an unsafe state
Deadlock Avoidance Algorithms

- Single-instance resources: Resource-allocation-graph algorithm
  - Claim edge Ti->Rj: Process Ti may request resource Rj
  - Claim edges change to request edges upon request and from assignment edges upon release
  - Upon request: Ensure no cycle will be created by allocation
- Multiple-instance resources: Banker’s algorithm

Deadlock Detection

- Detect deadlock after it happens
- System examines state from time-to-time
  - E.g.: analyze the resource allocation graph => Can collapse to wait-for graph of threads
- How often to detect deadlocks?
  - Upon each request?
  - Periodically?
  - Based on some thresholds?

Deadlock Recovery

- Process termination
  - Kill all processes in the deadlock cycle
  - Kill a process holding resource:
    - What happens to state of shared resources?
    - Which thread to select?
- Resource preemption
  - Take away resource forcibly
  - Rollback to a safe state

Ostrich Algorithm

- Ignore potential deadlocks
- In many complex systems:
  - Detecting or avoiding deadlocks may be costly or inefficient
  - Deadlocks may occur very rarely
- Potential deadlocks exist in most OS code
  - E.g.: Unix, Windows
  - Typically handled by rebooting/killing threads, etc.
Related Problems

- Livelock
- Priority Inversion

Livelock

- Threads not blocked, but no progress
  - Continuously doing some activity, but state does not advance
- Examples:
  - 2 threads repeatedly backing off and trying again to grab locks
  - Network protocols: Two machines on LAN repeatedly trying to send data at the same time
- Solution: Break synchrony
  - Add randomization
  - E.g.: Exponential backoff for Ethernet protocol

Priority Inversion

- A higher priority process waiting for a lower priority process
- Consider 3 processes H, M, L in priority order
  - L holds lock, H waiting on lock
  - M pre-empts L => H has to wait due to M
- Priority Inheritance
  - Lower priority process holding a lock inherits higher priority process’s priority temporarily