Deadlocks

- A situation where nobody can make progress
- Examples:
  - 4-way stop sign
  - Others?

Outline

- Deadlocks
  - Definition
  - Deadlock Conditions
  - Deadlock Handling and Prevention

Deadlock Example

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

- 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
- How do we synchronize the philosophers?

**Deadlock 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)

**Deadlock Prevention**
- 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

**Single-Lock**
- Use the same lock to protect any critical section
  - Which deadlock condition does it avoid?
- Problems:
  - Too coarse-grained, restricts concurrency
  - May result in poor resource utilization or starvation
  - All-at-once: Acquire all resources in a single atomic operation
    - Difficult to implement
    - Reduces to single-lock approach
Backoff

```c
lock(mut1);
if (trylock(mut2)==-1)
    unlock(mut1);
```

- Use trylocks to avoid blocking
- Release resources if trylock fails
- Ensures that thread either holds all locks or none
- Which condition does it avoid?

Total Lock Ordering

- Thread A:
  ```c
  lock(mut1);
  lock(mut2);
  ```
- Thread B:
  ```c
  lock(mut1);
  lock(mut2);
  ```
- Acquire the locks in a fixed order
  - E.g.: Each thread acquires mut1 before mut2
  - Which condition does it avoid?

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 processes, etc.