Announcements

• Milestone 2 meetings in progress
• No lecture next week (Thanksgiving break)

Recap

• Group communication
  – Msg send tied to group membership
  – Views
  – Virtual synchrony
  – Atomic multicast
• Distributed algorithms:
  – Election
  – Mutual Exclusion

Outline for Today

• Finish discussion of distributed mutual exclusion
• Distributed Transactions
  – Local transactions
  – Distributed protocols

Critical Sections

• Three criteria of critical sections
  – Mutual exclusion
    • “One at a time”
  – Progress
    • “May enter if empty”
  – Bounded Waiting
    • “Will enter eventually”
• Will look at three algorithms for mutual exclusion in distributed systems

Centralized ME w/ Coordinator
Distributed ME w/ Lamport Clocks

ME using Ring + Token

- Circulating Token
- Token holder may enter critical section

Comparison of ME algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Distributed</td>
<td>2 ((n - 1))</td>
<td>2 ((n - 1))</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to (\infty)</td>
<td>0 to (n - 1)</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>

- Question: which is the most scalable? Fault-tolerant? Efficient?

Introduction to Transactions

- Transaction model:
  - BEGIN_TRANSACTION, END_TRANSACTION, ABORT_TRANSACTION, READ+WRITE
- ACID property
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity

- "all-or-nothing" property
- Outside observer either sees results in entirety, or see only old values

Example

```
BEGIN_TRANSACTION
reserve WP -> JFK;
reserve JFK -> Nairobi;
reserve Nairobi -> Malindi;
END_TRANSACTION
```

Consistency

- Invariants holds before transaction, and after
- Example: Journaling Filesystem

```java
// Invariant: all used inodes are referred to by directory entries
CreateFile(Directory dir, File filename)
BEGIN_TRANSACTION
Inode inode = freeNodes.getNewInode();
// invariant temporarily violated
dir.addentry(filename, inode);
END_TRANSACTION
// Invariant restored
```
Isolation

- Concurrent transactions do not interfere with each other
- Equivalent to Serializability:
  - for multiple, concurrent transactions, the end result is the same as if transaction were executed in some serial order
  - schedule of operations is important

Serializability

### Schedule 1

\[
\begin{align*}
& x = 0; \\
& x = x + 1; \\
& x = 0; \\
& x = x + 2; \\
& x = x + 3; \\
& x = 0;
\end{align*}
\]

Legal

### Schedule 2

\[
\begin{align*}
& x = 0; \\
& x = x + 1; \\
& x = 0; \\
& x = x + 2; \\
& x = x + 3; \\
& x = 0;
\end{align*}
\]

Legal

### Schedule 3

\[
\begin{align*}
& x = 0; \\
& x = x + 1; \\
& x = 0; \\
& x = x + 2; \\
& x = 0; \\
& x = x + 3;
\end{align*}
\]

Illegal

Durability

- If transaction commits, results are permanent
  - typically implemented by writing to stable or persistent storage

Scheduler

- Transactions
  - READWRITE
    - Transaction manager
  - BEGIN_TRANSACTION
  - END_TRANSACTION
  - LOCK
  - RELEASE
  - Data manager
  - Timestamp operations
  - Execute read/write

Summary

- Transactions
  - ACID

Thursday:
- heterogenous object migration (Emerald)
- virtual machine migration (VMware)