Multi-Object Synchronization
Main Points

• Problems with synchronizing multiple objects
• Definition of deadlock
  – Circular waiting for resources
• Conditions for its occurrence
• Solutions for avoiding and breaking deadlock
Large Programs

• What happens when we try to synchronize across multiple objects in a large program?
  – Each object with its own lock, condition variables
  – Is concurrency modular?

• Deadlock

• Performance

• Semantics/correctness
**Deadlock Definition**

- **Resource**: any (passive) thing needed by a thread to do its job (CPU, disk space, memory, lock)
  - Preemptable: can be taken away by OS
  - Non-preemptable: must leave with thread
- **Starvation**: thread waits indefinitely
- **Deadlock**: circular waiting for resources
  - Deadlock => starvation, but not vice versa
**Example: two locks**

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock1.acquire();</td>
<td>lock2.acquire();</td>
</tr>
<tr>
<td>lock2.acquire();</td>
<td>lock1.acquire();</td>
</tr>
<tr>
<td>lock2.release();</td>
<td>lock1.release();</td>
</tr>
<tr>
<td>lock1.release();</td>
<td>lock2.release();</td>
</tr>
</tbody>
</table>
Bidirectional Bounded Buffer

Thread A

buffer1.put(data);
buffer1.put(data);
buffer2.get();
buffer2.get();

Thread B

buffer2.put(data);
buffer2.put(data);
Buffer1.get();
Buffer1.get();
Two locks and a condition variable

Thread A

lock1.acquire();
...
lock2.acquire();
while (need to wait)
  condition.wait(lock2);
lock2.release();
...
lock1.release();

Thread B

lock1.acquire();
...
lock2.acquire();
....
condition.signal(lock2);
lock2.release();
...
lock1.release();
Yet another Example
Dining Lawyers

Each lawyer needs two chopsticks to eat. Each grabs chopstick on the right first.
Conditions for Deadlock

- Limited access to resources
  - If infinite resources, no deadlock!
- No preemption
  - If resources are virtual, can break deadlock
- Multiple independent requests
  - “wait while holding”
- Circular chain of requests
Circular Waiting

Thread A
- owned by
  - owned by
  - waiting for

Thread B
- waiting for

Y
- owned by

X
Solution #1: Detect and Fix

• Algorithm
  – Scan wait for graph
  – Detect cycles
  – Fix cycles

• How?
  – Remove one thread, reassign its resources
    • Requires exception handling code to be very robust
  – Roll back actions of one thread
    • Databases: all actions are provisional until committed
Solution #2: Deadlock Prevention

Eliminate one of the four conditions for deadlock

• Lock ordering
  – Always acquire locks in the same order
  – Example: move file from one directory to another
  – Widely used in OS kernels

• Design system to release resources and retry if need to wait
  – No “wait while holding”
  – Example: telephone circuit setup

• Infinite resources?
  – Ex: UNIX reserves a process for the sysadmin to run “kill”

• Acquire all needed resources in advance
Solution #3: Banker’s Algorithm

• Banker’s algorithm
  – State maximum resource needs in advance
  – Allocate resources dynamically when resource is needed -- wait if granting request would lead to deadlock
  – Request can be granted if some sequential ordering of threads is deadlock free
Possible System States

Unsafe

Safe

Deadlock
Definitions

- **Safe state:**
  - For any possible sequence of future resource requests, it is possible to eventually grant all requests.
  - May require waiting even when resources are available!

- **Unsafe state:**
  - Some sequence of resource requests can result in deadlock.

- **Doomed state:**
  - All possible computations lead to deadlock.
Banker’s Algorithm

• Grant request iff result is a safe state
• Sum of maximum resource needs of current threads can be greater than the total resources
  – Provided there is some way for all the threads to finish without getting into deadlock
• Example: proceed iff
  – total available resources - # allocated >= max remaining that might be needed by this thread in order to finish
  – Guarantees this thread can finish
Lock-Free Data Structures

• Assume compare and swap atomic instruction
  – Limitation: swap a single memory location
  – Only supported on some processor architectures

• Rewrite critical section
  – Create copy of data structure
  – Modify copy
  – Swap in pointer to copy iff no one else has
  – Restart if pointer has changed
Lock-Free Bounded Buffer

get() {
    do {
        mine = ConsistentCopy(p);
        if (mine.front == mine.last)
            mine.queue.Add(self);
        else {
            item = mine.buf[
                mine.front % size];
            mine.front++;
        }
    } while ((compare&swap(mine, p) != p);
    wake up waiter if needed
    return item.
}