Synchronization

Heechul Yun
Recap: Monitor

- Monitor
  - A lock (mutual exclusion) + condition variables (scheduling)
  - Some languages like Java natively support this, but you can use monitors in other languages like C/C++

- Lock: mutual exclusion
  - Protects the shared data structures inside the monitor
  - Always acquire it to enter the monitor
  - Always release it to leave the monitor

- Condition Variable: scheduling
  - Allow thread to wait on certain events inside the monitor
  - Key idea: to wait (sleep) inside the monitor, it first releases the lock and go to sleep atomically
Agenda

• Famous Synchronization Bugs
  – THERAC-25
  – Mars Pathfinder
Therac 25

- Computer controlled medical X-ray treatments
- Six people died/injured due to massive overdoses (1985-1987)

Image source: [http://idg.bg/test/cwd/2008/7/14/21367-radiation_therapy.JPG](http://idg.bg/test/cwd/2008/7/14/21367-radiation_therapy.JPG)
## Accident History

<table>
<thead>
<tr>
<th>Date</th>
<th>What happened</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 1985</td>
<td>First overdose</td>
</tr>
<tr>
<td>July-Dec 1985</td>
<td>2nd and 3rd overdoses. Lawsuit against the manufacturer and hospital</td>
</tr>
<tr>
<td>Jan-Feb 1986</td>
<td>Manufacturer denied the possibility of overdoses</td>
</tr>
<tr>
<td>Mar-Apr 1986</td>
<td>Two more overdoses</td>
</tr>
<tr>
<td>May-Dec 1986</td>
<td>FDA orders corrective action plans to the manufacturer</td>
</tr>
<tr>
<td>Jan 1987</td>
<td>Sixth overdose</td>
</tr>
<tr>
<td>Nov 1988</td>
<td>Final safety analysis report</td>
</tr>
</tbody>
</table>
The Problem

- X-ray must be dosed with the filter in place
- But sometimes, X-ray was dosed w/o the filter

The Bug

unsigned char in_progress = 1;

Thread 1: // tray movement thread (periodic)
   if (system_is_ready())
      in_progress = 0;
   else
      in_progress++;

Thread 2: // X-ray control thread.
   if (in_progress == 0)
      start_radiation();

• Can you spot the bug?
unsigned char in_progress;

Thread 1 : // tray movement thread (periodic)
    if (system_is_ready())
        in_progress = 0;
    else
        in_progress = 1;

Thread 2 : // X-ray control thread.
    if (in_progress == 0)
        start_radiation();

• Can you do better using a monitor?
Monitor Version

Mutex lock;
Condition ready;
unsigned char in_progress;

Thread 1: // on finishing tray movement
  lock.acquire();
  in_progress = 0;
  ready.signal();
  lock.release();

Thread 2: // X-ray control thread.
  lock.acquire();
  while (in_progress)
    ready.wait(&lock);
  start_radiation();
  lock.release();

• No periodic check is needed.
Mars Pathfinder

- Landed on Mars, July 4, 1997
- After operating for a while, it rebooted itself
The Bug

• Three threads with priorities
  – Weather data thread (low priority)
  – Communication thread (medium priority)
  – Information bus thread (high priority)

• Each thread obtains a lock to write data on the shared memory

• High priority thread can’t acquire the lock for a very long time → something must be wrong. Let’s reboot!
Priority Inversion

- High priority thread is delayed by the medium priority thread (potentially) indefinitely!!!
Solution

• Priority inheritance protocol [Sha’90]
  – If a high priority thread is waiting on a lock, boost the priority of the lock owner thread (low priority) to that of the high priority thread.

• Remotely patched the code
  – To use the priority inheritance protocol in the lock
  – First-ever(?) interplanetary remote debugging

Priority Inheritance

Old

High

Medium

Low

New

High

Medium

Low

lock()

unlock()

blocked

lock() unlock()

Boost priority

lock() unlock()
Summary

• Race condition
  – A situation when two or more threads read and write shared data at the same time

• Critical section
  – Code sections of potential race conditions

• Mutual exclusion
  – If a thread executes its critical section, no other threads can enter their critical sections

• Peterson’s solution
  – Software only solution providing mutual exclusion
Summary

• Spinlock
  – Spin on waiting
  – Use synchronization instructions (test&set)

• Mutex
  – Sleep on waiting

• Semaphore
  – Powerful tool, but often difficult to use

• Monitor
  – Powerful and (relatively) easy to use
Mutex lock;
Condition full, empty;

produce (item)
{
    __________
    while (queue.isFull())
        empty.wait(&lock);
    queue.enqueue(item);
    full.signal();
    __________
}

consume()
{
    __________
    while (queue.isEmpty())
        __________
        item = queue.dequeue(item);
        __________
        __________
        return item;
    __________
}

Semaphore mutex = 1, full = 0, empty = N;

produce (item)
{
    __________;
    P(&mutex);
    queue.enqueue(item);
    V(&mutex);
    __________;
}

consume()
{
    __________;
    P(&mutex);
    item = queue.dequeue();
    V(&mutex);
    __________;
    return item;
}
Mutex lock;
Condition full, empty;

produce (item)
{
    lock.acquire();
    while (queue.isFull())
        empty.wait(&lock);
    queue.enqueue(item);
    full.signal();
    lock.release();
}

consume()
{
    lock.acquire();
    while (queue.isEmpty())
        full.wait(&lock);
    item = queue.dequeue(item);
    empty.signal();
    lock.release();
    return item;
}