Mutual Exclusion Using Load/Store

A protocol based on two shared variables $c1$ and $c2$. Initially, both $c1$ and $c2$ are 0 (not busy).

Process 1

```
...  c1=1;
L: if c2=1 then go to L
    < critical section>
c1=0;
```

Process 2

```
...  c2=1;
L: if c1=1 then go to L
    < critical section>
c2=0;
```

What is wrong?

Mutual Exclusion: second attempt

To avoid deadlock, let a process give up the reservation (i.e. Process 1 sets $c1$ to 0) while waiting.

Process 1

```
L:  c1=1;
    if c2=1 & turn=1 
    { c1=0; go to L } 
    < critical section>
    c1=0
```

Process 2

```
L:  c2=1;
    if c1=1 then 
    { c2=0; go to L } 
    < critical section>
    c2=0
```

• Deadlock is not possible but with a low probability a livelock may occur.
• An unlucky process may never get to enter the critical section → starvation

A Protocol for Mutual Exclusion

T. Dekker, 1966

A protocol based on 3 shared variables $c1$, $c2$ and turn. Initially, both $c1$ and $c2$ are 0 (not busy).

Process 1

```
c1=1;
    turn = 1;
L: if c2=1 & turn=1 
    then go to L 
    < critical section>
c1=0;
```

Process 2

```
c2=1;
    turn = 2;
L: if c1=1 & turn=2 
    then go to L 
    < critical section>
c2=0;
```

• $\text{turn} = i$ ensures that only process $i$ can wait
• variables $c1$ and $c2$ ensure mutual exclusion

Solution for $n$ processes was given by Dijkstra and is quite tricky!

Analysis of Dekker’s Algorithm

Scenario 1

```
...  c1=1;
      turn = 1;
L: if c2=1 & turn=1 
    then go to L 
    < critical section>
    c1=0;
```

Scenario 2

```
...  c2=1;
      turn = 2;
L: if c1=1 & turn=2 
    then go to L 
    < critical section>
    c2=0;
```

Last time...

• Implementations for semaphores
  – Test&set
  – Compare&swap
  – Load-reserve & store-conditional
• Sequential consistency vs. weaker consistencies
  – Agreement between hardware and software
  – For weaker consistency models, hardware provides extra instructions for software to implement stronger guarantees, e.g., memory fences
N-process Mutual Exclusion
Lamport’s Bakery Algorithm

Process i

choosing[i] = 1;
num[i] = max(num[0], ..., num[N-1]) + 1;
choosing[i] = 0;

for(j = 0; j < N; j++)
{
    while(choosing[j]);
    while(num[j] &&
        ((num[j] < num[i]) ||
         (num[j] == num[i] && j < i)));
}

num[i] = 0;

Entry Code
num[j] = 0, for all j

Exit Code
num[i] = 0;

CSE 490/590, Spring 2011

Memory Coherence in SMPs

Suppose CPU-1 updates A to 200.
write-back: memory and cache-2 have stale values
write-through: cache-2 has a stale value

Do these stale values matter?
What is the view of shared memory for programming?

CSE 490/590, Spring 2011

Write-back Caches & SC

• T1 is executed

cache-1 writes back Y

• T2 executed

cache-1 writes back X

cache-2 writes back X’ & Y’

Write-through Caches & SC

• T1 executed

Write-through caches don’t preserve sequential consistency either

• T2 executed

Cache Coherence vs. Memory Consistency

• A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors
  – i.e., updates are not lost

• A memory consistency model gives the rules on when a write by one processor can be observed by a read on another
  – Equivalently, what values can be seen by a load

• A cache coherence protocol is not enough to ensure sequential consistency
  – But if sequentially consistent, then caches must be coherent

• Combination of cache coherence protocol plus processor memory reorder buffer implements a given machine’s memory consistency model

CSE 490/590 Administrivia

• CSE Graduate Conference on Friday, 4/15 @ 145 Student Union
  – No class

• Keyboards available for pickup at my office

• Updated project 2 with more clarifications & grading criteria

CSE 490/590, Spring 2011

Memory Coherence in SMPs

Write-through Caches & SC

Cache Coherence vs. Memory Consistency

CSE 490/590, Spring 2011
Maintaining Cache Coherence

Hardware support is required such that
• only one processor at a time has write permission for a location
• no processor can load a stale copy of the location after a write
⇒ cache coherence protocols

Warmup: Parallel I/O

Either Cache or DMA can be the Bus Master and effect transfers

Problems with Parallel I/O

Memory → Disk: Physical memory may be stale if cache copy is dirty
Disk → Memory: Cache may hold stale data and not see memory writes

Acknowledgements

• These slides heavily contain material developed and copyright by
  – Krste Asanovic (MIT/UCB)
  – David Patterson (UCB)
• And also by:
  – Arvind (MIT)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
• MIT material derived from course 6.823
• UCB material derived from course CS252