

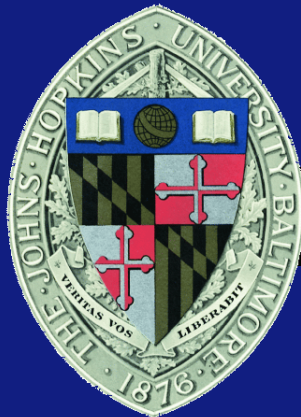
Lecture 13.2

Read, Modify, Write Atomics

EN 600.320/420

Instructor: Randal Burns

15 March 2017



Department of Computer Science, *Johns Hopkins University*

The Next Layer of Concepts

- Variants on number of processes
 - Infinitely many processes
 - Sparse process id address space (symmetric algs.)
- Spinning on local registers only
- For different memory models
 - CC, DSM



Building on Primitives

- Common atomic operations (building blocks):
 - Read
 - Write
 - Test-and-set
 - Swap
 - Fetch and add (fetch and increment)
 - Read-modify-write
 - Compare-and-swap



Test-and-Set Bit

- Two operations
 - Reset: write 0
 - Test and set: write 1 and return old value
- Trivial deadlock free synchronization

```
await (test-and-set(x) = 0);  
critical section  
reset(x);
```

- This is called a *spin lock*
 - Mutual exclusion, deadlock free
 - Not starvation resistant



Test-and-Test-and-Set Bit

- Test-and-set alg. writes bit every iteration
 - Invalidates caches even when data don't change
- Test-and-test-and-set
 - Supports test w/out set
- Produces fewer cache misses
 - What's the miss pattern during contention

```
await (x=0);  
while (test-and-set(x) = 1) do  
    await (x=0) od;  
critical section  
reset(x);
```



What's wrong with Spin Locks?

- Every process spins on shared state
- When lock is freed, all processes attempt to acquire
- Performance varies with contention:
 - Low contention good (simple algorithms)
 - High contention bad (burst of activity: messages and cache invalidations)
- Can be addressed with backoff policies
 - Like exponential backoff in TPC
- But, queuing is better



Ticket Algorithm

- Bakery algorithms using read-modify-write
 - \langle and \rangle indicate RMW boundaries

THE TICKET ALGORITHM: process i 's program.

```
constant  $N = \{0, 1, \dots, n - 1\}$ 
shared  $(ticket, valid)$  a read-modify-write register ranges over  $N \times N$ ;
  initially  $ticket = valid$ 
local  $(ticket_i, valid_i)$  ranges over  $N \times N$ 

1   $\langle (ticket_i, valid_i) := (ticket, valid);$ 
2     $ticket := (ticket + 1) \bmod n \rangle;$ 
3  while  $ticket_i \neq valid_i$  do
4     $\langle valid_i := valid \rangle$  od;
5  critical section;
6   $\langle valid := (valid + 1) \bmod n \rangle;$ 
```



Properties of RMW Ticket Alg.

- FIFO: in the order of successful RMW
- Mutual exclusion and deadlock freedom
- Uses one shared register that holds n^2 values
- This is the power of H/W support
 - Modern processors provide some variant of RMW



Waiting w/out the Busy Wait

- The Semaphore S
 - $up(S)$ increase the value of S
 - $down(S)$ decrease the value of S
 - Binary semaphore takes values 0 and 1
- Using the semaphore
 - $down(S)$; critical section; $up(S)$;
 - To realize deadlock-free, mutual exclusion
- Where does the busy wait go?
 - Nowhere: implement semaphores with test-and-set
 - Into the kernel: one process does all the busy waiting
 - Into hardware: use interrupts

