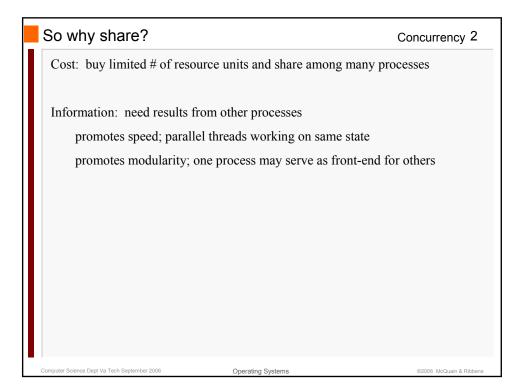
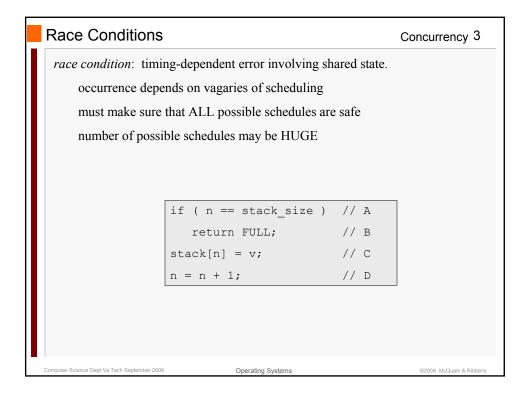
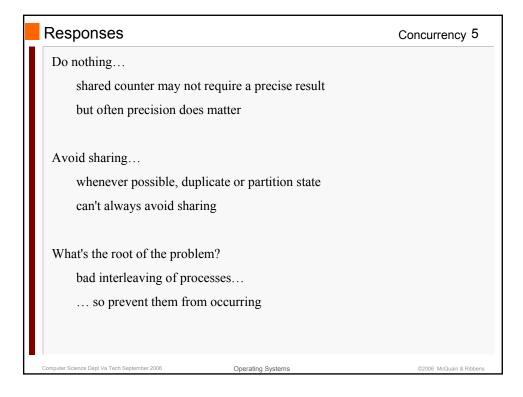
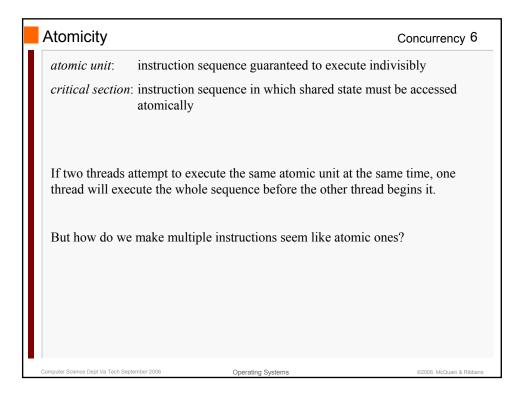
Multiple Processes in One World	Concurrency 1
Safe? Not automatically.	
Suppose one process is modifying shared state information.	
Can another process safely modify at the same time?	
Can another process safely read at the same time?	
Are there inherently safe sharing scenarios?	
Errors may be irreproducible.	
Consider a shared counter	
Computer Science Dept Va Tech September 2006 Operating Systems	©2006 McQuain & Ribbens



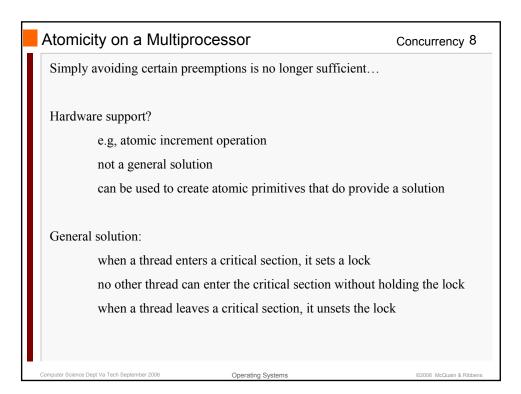


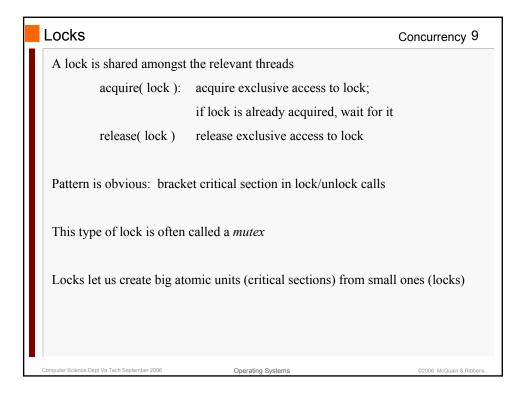
Race Conditions				Concurrency 4
	<pre>// Thread a i = 0; while (i < 10) i = i + 1; print "A won!";</pre>		<pre>// Thread b i = 0; while (i > -10) i = i - 1; print "B won!";</pre>	
Note: var Who wins Must one				
Computer Science Dept Va	a Tech September 2006 O	perating S	ystems	©2006 McQuain & Ribbens

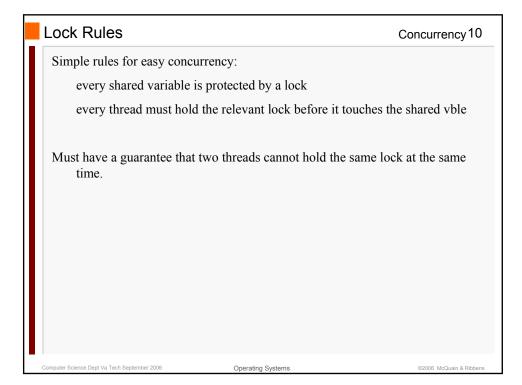




Atomicity on a UniprocessorConcurrency 7The only requirement is that when a thread is in a critical section it will not be preempted.OS traditional approach is that threads will disable/enable interrupts. $old = intr_disable();$ hits = hits + 1; $intr_set_level(old);$ Dangerous... programmer error can disrupt entire system, not merely lead to icorrect results for the threads that are sharing state.Could have the scheduler check the thread's PC versus a table of critical section addresses... requires compiler support and extra work.







Implementing Locks

A simple attempt:

```
acquire( Lock& L ) {
   while ( L == 0 ) continue;
   L = 0;
}
release( Lock& L ) {
   L = 1;
}
```



Concurrency 11

Focus or		
	acquire(Lock& L) {	
	<pre>disable_preemption();</pre>	
	while (L == 0) continue;	
	L = 0;	
	<pre>enable_preemption();</pre>	
	}	
	release(Lock& L) {	
	L = 1;	
	}	

