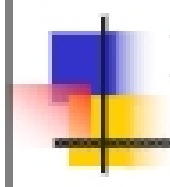
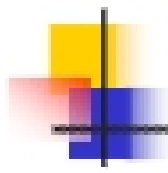


Implementing Mutual Exclusion

Arvind Krishnamurthy

Spring 2004



Disable Interrupts

- Uniprocessor only: an operation will be atomic as long as a context switch does not occur
- Two ways for dispatcher to get control:
  - Internal events – thread does something to relinquish the CPU
  - External events – interrupts cause dispatcher to take CPU away
- Need to prevent both internal and external events
  - Preventing internal events is easy
  - Preventing external events require disabling interrupts
    - Hardware delays handling of external events

Using interrupts (1)

- Flawed but simple:

```
Lock::Acquire()   { disable interrupts; }

Lock::Release()   { enable interrupts; }
```

- Problems:
  - the above may make some user thread never give back CPU
  - critical sections can be arbitrarily long --- it may take too long to respond to an interrupt --- real-time system won't be happy
  - this won't work for other higher-level primitives such as semaphores and condition variables

Using interrupts (2)

Key idea: maintain a lock variable and impose mutual exclusion only on the operations of testing and setting that variable

```
class Lock { int value = FREE; }

Lock::Acquire()  {
   Disable interrupts;
   while (value != FREE) {
     Enable interrupts;
     Disable interrupts;
    }
    value = BUSY;
    Enable interrupts;
}
```

```
Lock::Release()  {
   Disable interrupts;
   value = FREE;
   Enable interrupts;
}
```

Using interrupts (3)

Key idea: Use a queue to maintain a list of threads waiting for the lock. Avoid busy-waiting:

```
class Lock { int value = FREE; }

Lock::Acquire() {
   Disable interrupts;
   if (value == BUSY) {
      Put on queue of threads waiting for lock;
      Go to sleep;
                // Enable interrupts ? No!
   } else {
      value = BUSY;
   }
   Enable interrupts
}
```

```
Lock::Release() {
   Disable interrupts;
   if anyone on wait queue {
      Take a waiting thread off wait
      queue and put it at the front
      of the ready queue;
   } else {
      value = FREE;
   }
   Enable interrupts
}
```

When to re-enable interrupts?

- Before putting the thread on the wait queue ?
  - Then Release can check the queue, and not wake the thread up
- After putting the thread on the wait queue but before going to sleep
  - Then Release puts the thread on the ready queue, but the thread still thinks it needs to go to sleep!
  - It will go to sleep and miss the wakeup from Release
- In Nachos, interrupts are disabled when you call Thread:Sleep; it is the responsibility of the next thread-to-run to re-enable interrupts.