Two Versions of Semaphores

■ Semaphores from last time (simplified):

wait (s): signal (s): s = s - 1s = s + 1if (s < 0)if $(s \le 0)$

block the thread wake up one of that called wait(s) the waiting threads

otherwise continue into CS

■ "Classical" version of semaphores:

wait (s): signal (s): if $(s \le 0)$

if (a thread is waiting) block the thread wake up one of that called wait(s) the waiting threads

s = s - 1s = s + 1

continue into CS

■ Do both work? What is the difference??

Implementing Semaphores

■ Implementing semaphores using busywaiting:

wait (s): signal (s): while $(s \le 0)$ s = s + 1do nothing; s = s - 1

- Evaluation:
 - ✗ Waiting threads wastes time busy-waiting (doing nothing useful, wasting CPU time)
 - X The code inside wait(s) and signal(s) is a critical section also, and it's not protected
 - X Doesn't support a queue of multiple blocked threads waiting on the semaphore (why is this bad?)

Fall 2002, Lecture 12

Fall 2002, Lecture 12

Implementing Semaphores (cont.)

■ Implementing semaphores (not fully) by disabling interrupts:

wait (s): signal (s):

disable interrupts disable interrupts

while $(s \le 0)$ s = s + 1

do nothing;

s = s - 1

enable interrupts enable interrupts

- Evaluation:
 - ✔ Protects code inside wait(s) and signal(s)
 - ✗ Waiting threads wastes time busy-waiting
 - ✗ Doesn't support queue of blocked threads waiting on the semaphore
 - ✗ Users can't disable interrupts
 - X Can interfere with timer, which might be needed by other applications
 - Doesn't work on multiprocessors

Implementing Semaphores (cont.)

■ Implementing semaphores (not fully) using a test&set instruction.

wait (s): signal (s):

while (test&set(lk)!=0) while (test&set(lk)!=0)

do nothing: do nothing:

while $(s \le 0)$ s = s + 1

do nothing; s = s - 1

lk = 0lk = 0

- Operation:
 - Lock "lk" has an initial value of 0
 - If "lk" is free (lk=0), test&set atomically:
 - reads 0, sets value to 1, and returns 0
 - loop test fails, meaning lock is now busy
 - If "lk" is busy (lk=1), test&set atomically:
 - reads 1, sets value to 1, and returns 1
 - loop test is true, so loop continues until someone releases the lock

Fall 2002, Lecture 12

Fall 2002, Lecture 12

Implementing Semaphores (cont.)

- Test&set is an example of an atomic read-modify-write (RMW) instruction
 - RMW instructions <u>atomically</u> read a value from memory, modify it, and write the new value to memory
 - Test&set on most CPUs
 - Exchange Intel x86 swaps values between register and memory
 - Compare&swap Motorola 68xxx read value, if value matches value in register r1, exchange register r1 and value
- **■** Evaluation:
 - Can be made to work, even on multiprocessors (although there may be some cache consistency problems)
 - ✗ Waiting threads wastes time busy-waiting
 - ✗ Doesn't support queue of blocked threads waiting on the semaphore

Fall 2002, Lecture 12

Semaphores in Nachos

- The class Semaphore is defined in threads/synch.h and synch.cc
 - The classes Lock and Condition are also defined, but their member functions are empty (implementation left as exercise)
- Interesting functions:
 - Semaphores:
 - Semaphore::Semaphore() creates a semaphore with specified name & value
 - Semaphore::P() semaphore wait
 - Semaphore::V() semaphore signal
 - Locks:
 - Lock::Acquire()
 - Lock::Release()
 - Condition variables:
 - Condition::Wait()
 - Condition::Signal()

Fall 2002, Lecture 12

Semaphores in Nachos

```
void
Semaphore::P()
  IntStatus oldLevel = interrupt->
     SetLevel(IntOff); // disable interrupts
  while (value == 0) {
                           // sema not avail
     queue->
                           // so go to sleep
        Append((void *)currentThread);
     currentThread->Sleep();
  }
                    // semaphore available,
  value--;
                    // consume its value
  (void) interrupt->
                      // re-enable interrupts
     SetLevel(oldLevel);
```

Semaphores in Nachos (cont.)

Fall 2002, Lecture 12 8 Fall 2002, Lecture 12