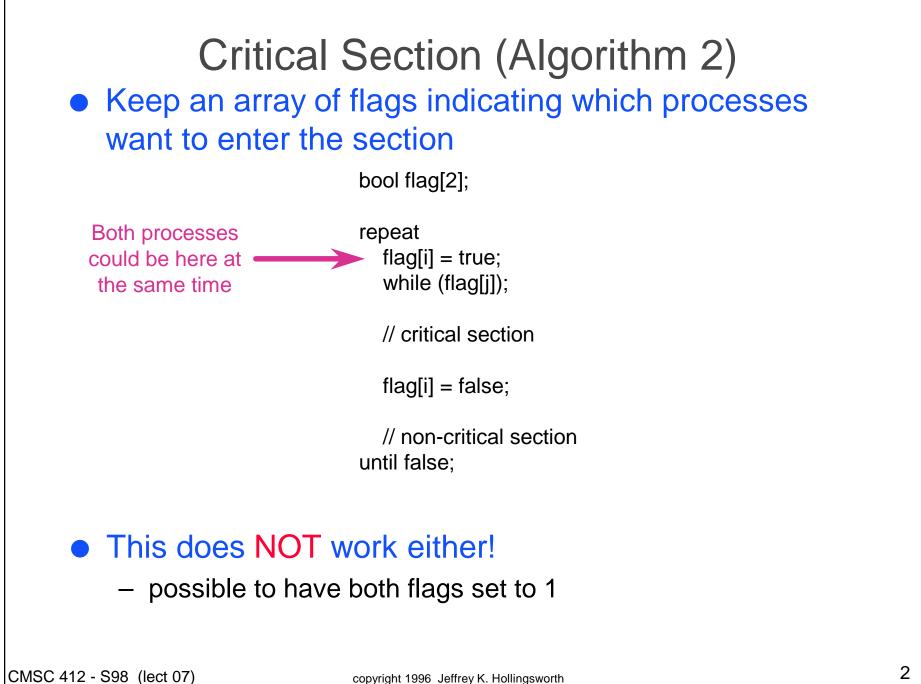
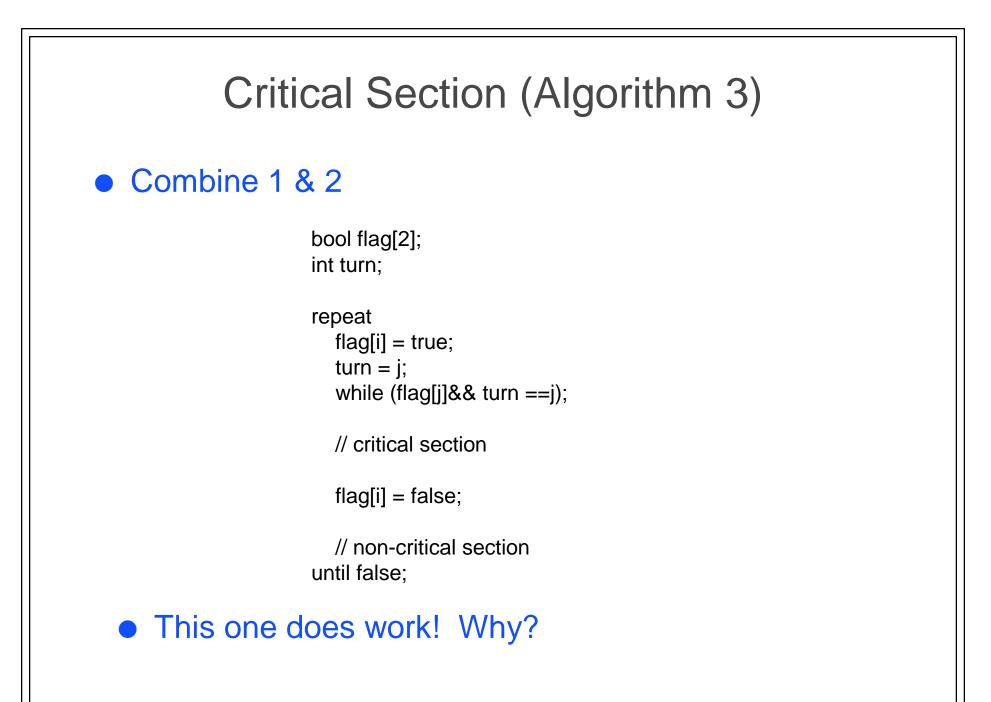
Announcements

- Reading chapter 6 (6.4 and 6.5)
- Midterm #1 is March 5 in class
- Late Policy for programs
 - no late work will be accepted
 - illness and family emergency will be considered on a case by case basis





CMSC 412 - S98 (lect 07)

Critical Section (many processes)

```
What if we have several processes?
One option is the Bakery algorithm
bool choosing[n];
integer number[n];
```

```
choosing[i] = true;
number[i] = max(number[0],..number[n-1])+1;
choosing[i] = false;
for j = 0 to n-1
    while choosing[j];
    while number[j] != 0 and ((number[j], j) < number[i],i);
end
// critical section
number[i] = 0
```

Bakery Algorithm - explained

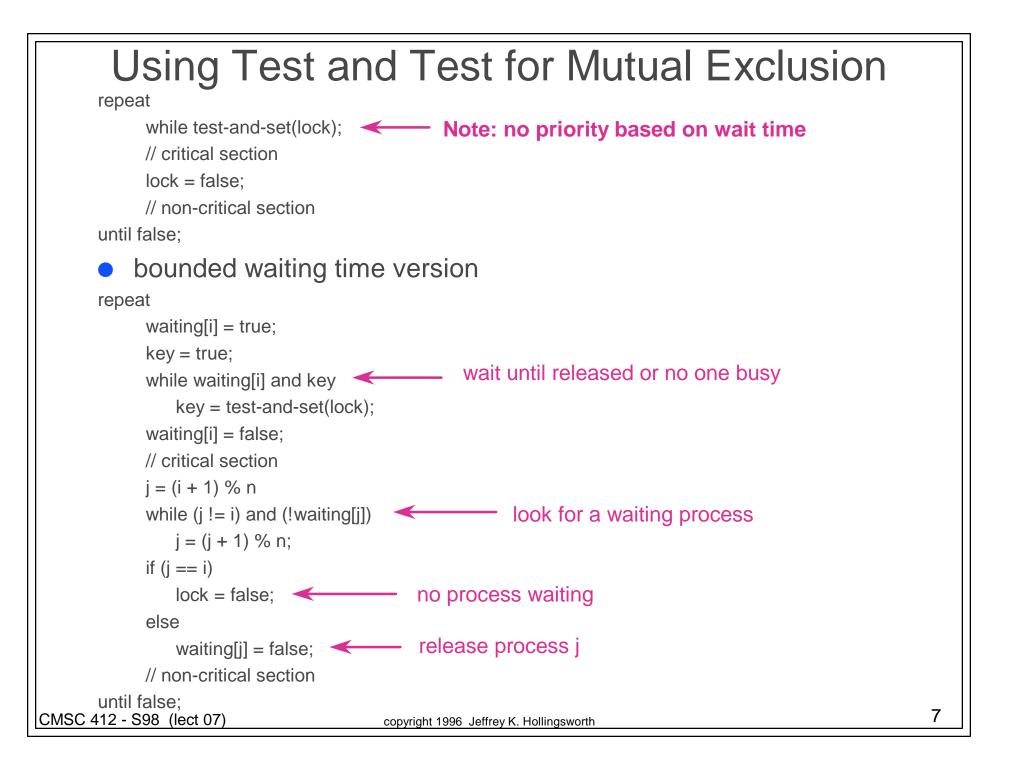
- When a process wants to enter critical section, it takes a number
 - however, assigning a unique number to each process is not possible
 - it requires a critical section!
 - however, to break ties we can used the lowest numbered process id
- Each process waits until its number is the highest one
 - it can then enter the critical section
- provides fairness since each process is served in the order they requested the critical section

Synchronization Hardware

- If it's hard to do synchronization in software, why not do it in hardware?
- Disable Interrupts
 - works, but is not a great idea since important events may be lost.
 - doesn't generalize to multi-processors
- test-and-set instruction
 - one atomic operation
 - executes without being interrupted
 - operates on one bit of memory
 - returns the previous value and sets the bit to one

swap instruction

- one atomic operation
- swap(a,b) puts the old value of b into a and of a into b



Semaphores

- getting critical section problem correct is difficult
 - harder to generalize to other synchronization problems
 - Alternative is semaphores

• semaphores

- integer variable
- only access is through atomic operations

```
• P (or wait)
```

```
while s \le 0;
```

```
s = s - 1;
```

• V (or signal)

```
s = s + 1
```

Using Semaphores

• critical section

repeat

P(mutex);

// critical section

V(mutex);

// non-critical section

until false;

Require that Process 2 begin statement S2 after Process 1 has completed statement S1:

Process 2 S1 V(synch) Process 1 P(synch) S2 Implementing semaphores

- Busy waiting implementations
- Instead of busy waiting, process can block itself
 - place process into queue associated with semaphore
 - state of process switched to waiting state
 - transfer control to CPU scheduler
 - process gets restarted when some other process executes a signal operations