ECE391
Computer System Engineering
Lecture 7

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

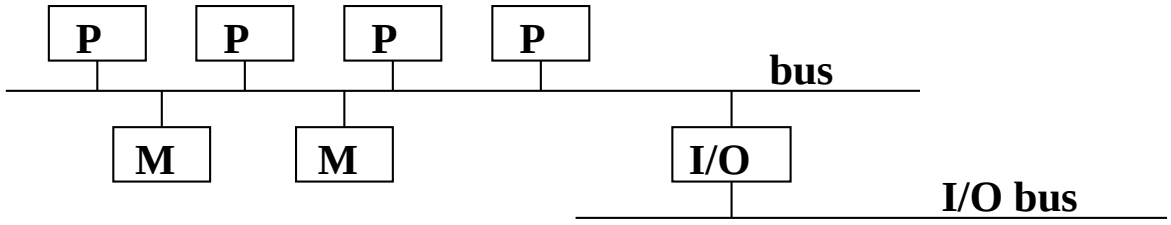
Fall 2022

Lecture Topics

- Multiprocessors and locks
- Spin locks

Multiprocessors and Locks

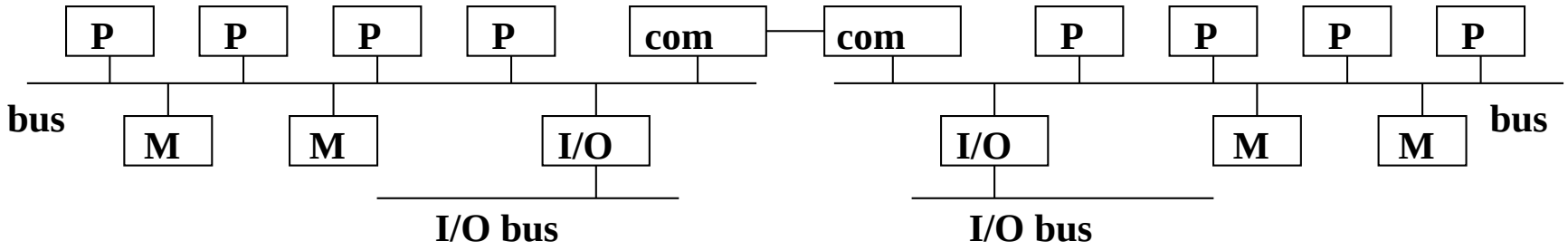
- We solved the critical section problem for uniprocessors
- What about multiprocessors?
  - *CLI … critical section …. STI*
- What is a multiprocessor?
  - usually a symmetric multiprocessor (SMP)



  - symmetric aspect: all processors have equal latency to all memory banks
  - multicore processors are similar from our perspective

Multiprocessors and Locks (cont.)

- Some non-uniform memory architecture (NUMA) machines were built

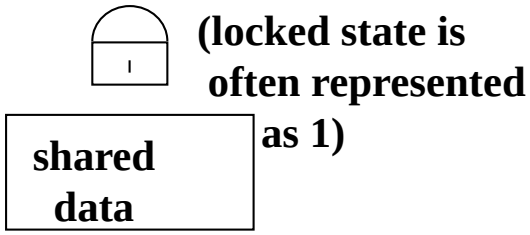


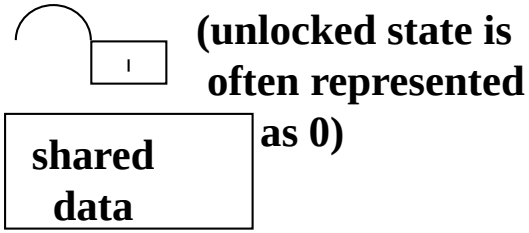
Multiprocessors and Locks (cont.)

- Multithreaded code is not protected by ***IF***
- Why haven't we solved the atomicity problem on multiprocessors?
  - interrupts are masked if ***IF*** is cleared!
  - answer: ***IF*** is not cleared on other processors!
  - just tell other processors to clear IF, too?
    - too slow
    - requires an interrupt!
- We need to use shared memory to synchronize…

Multiprocessors and Locks (cont.)

- Logically, we use a lock
  - when we want to access a piece of shared data we first look at the lock
  - if it's locked, we wait until it's unlocked

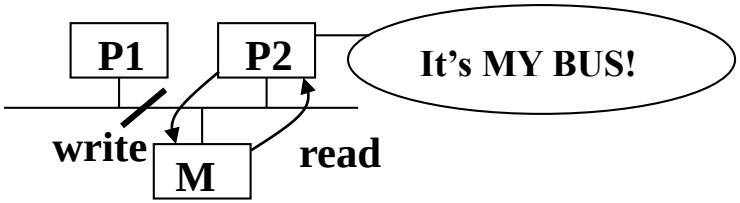




  - once it's unlocked
    - we lock it
    - access the data
    - then unlock it

Multiprocessors and Locks (cont.)

- Locking must be atomic with respect to other processors!



INCL (%EAX)

read (%EAX)
add 1
put it back

LOCK: Prefix - execute following instruction with bus locked

LOCK INCL (%EAX)

An Example of Locking Implementation

```
/* The caller defines int lock. Calling TestAndSet (&lock) sets lock to 1 and returns the old
value of lock. If lock is 0 then TestAndSet(&lock)returns 0 and sets the lock to 1. If the
lock is already set to 1  by another process or thread, then 1 is returned. The caller keeps
retrying TestAndSet(&lock) until it returns 0. */
```

TestAndSet:

```
……
pushl %ebx
movl  8(%ebp), %ebx        # &lock to ebx
movl  $1, %eax             # 1 (true) to eax

# Swap eax and lock, value 1(true) is copied to lock, eax receives old lock value
xchg  %eax, (%ebx)         #Atomically exchange eax with lock.
                           #The atomicity of xchg guarntees that at most
                           #one thread holds the lock at any point in time
popl  %ebx
……
ret                        #return value (old value of lock) is already in eax
```

xchg => exchanges the contents of a register with the contents of any other register or memory location

Spin Locks

- The simplest lock
  - spin lock
  - lock op

```
do {
    try to change lock variable from 0 to 1
} while (attempt failed);
```

  - other work to do? ignore it!
  - spin in a tight loop on the lock (hence the name)
- Once successful, program/interrupt handler owns the lock

Spin Locks (cont.)

- Only the owner can unlock
  - How?
    - (change lock variable to 0)
  - Need to be atomic?
    - (no, only owner can change when locked)

Linux Spin Lock API

- Static initialization

  static spinlock_t a_lock = SPIN_LOCK_UNLOCKED;

- Dynamic initialization

  spin_lock_init (&a_lock);

- When is dynamic initialization safe?
  - lock must not be in use (race condition!)
  - other synchronization method must prevent use

Linux Spin Lock API – Basic Functions

```
void spin_lock (spinlock_t* lock);

void spin_unlock (spinlock_t* lock);
```

Linux Spin Lock API – Testing Functions

```
int spin_is_locked (spinlock_t* lock);
```

returns 1 if held, 0 if not, but beware of races!

```
int spin_trylock (spinlock_t* lock);
```

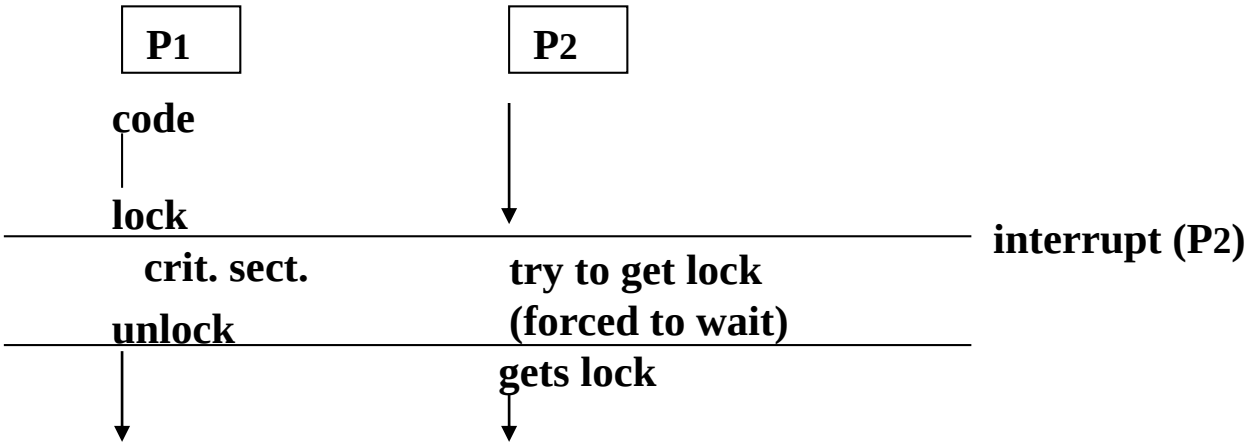
make one attempt; returns 1 on success, 0 on failure

```
void spin_unlock_wait (spinlock_t* lock);
```

wait until available (race condition again!)

Linux Spin Lock API (cont.)

- Is spinlock enough to protect a critical section?



Linux Spin Lock API (cont.)

- What about ?

- Still need CLI/STI
- Which is first, CLI or lock?
  - CLI first
  - interrupt may occur between them, leading to scenario above

Linux' Lock/CLI Combo

```
static spinlock_t the_lock = SPIN_LOCK_UNLOCKED;
unsigned long flags;

spin_lock_irqsave (&the_lock, flags);
/* the critical section */
spin_unlock_irqrestore (&the_lock, flags);
```

```
asm volatile ("
      PUSHFL
      POPL %0
      CLI
" : "=g" (flags)     /* outputs  */
  :                  /* inputs   */
  : "memory"         /* clobbers */
);
spin_lock (&the_lock);
```

Linux' Lock/CLI Combo (cont)

```
spin_unlock (&the_lock);
asm volatile ("
     PUSHL %0
     POPFL
" :                         /* outputs  */
  : "g" (flags)             /* inputs   */
  : "memory", "cc"          /* clobbers */
);
```

Comments on code

- Notice that *spin_lock_irqsave* changes the flags argument (it's a macro)
- The "memory" argument
  - tells compiler that all memory is written by assembly block
  - prevents compiler from moving memory ops across assembly
- The "cc" argument
  - condition codes change
  - can lead to subtle bugs if left out!
- *spin_lock* and *spin_unlock* calls
  - become NOPs on uniprocessors
  - in that case, calls just change IF
- Restore rationale: may have had IF=0 on entry; if so, STI is unsafe