Lecture Topics

- Shared resources
- Critical sections

MP1 Handin and Demo Schedule

- Code must be committed to master/main branch on GitLab by
  - 9:59AM on 2/18 (11:59PM on 2/18 for ZJUI)

- Handin Demo
  - Monday 2/22, Starts at 6 PM: All ZJUI Students and Last names from A to J
  - Tuesday 2/23, Starts at 6 PM: Last names from K to Z

Shared Data and Resources (1)

- The question
  - interrupt handlers and programs share resources
    - What resources are shared between them?
    - How might interactions cause problems?
    - What can we do to fix those problems?

Thought Problem on Shared Resources (2)

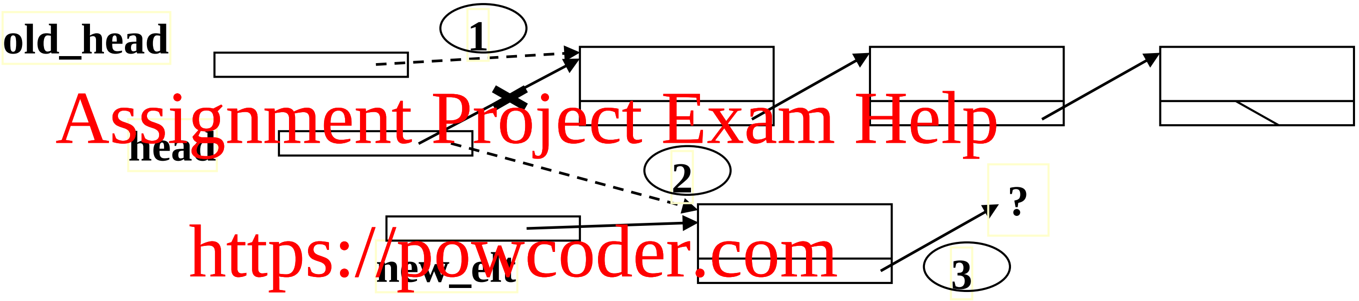
- Obvious things
  - registers
    - solution? save them to the stack
  - memory
    - solution? privatize
    - will still need to share some things; discussed later

Thought Problem on Shared Resources (3)

- Less obvious
  - condition codes
    - solution? again, save them to the stack
  - shared data
- More subtle
  - external state (e.g., on devices)
  - compiler optimization (e.g., volatility)
  - security leaks
    - e.g., application waits for interrupt, then observes values written by OS to stack
    - solution? use separate stack for kernel

Examples of Shared Resources:

Example #1: a shared linked list



step 1: **`old_head = head;`**

step 2: **`head = new_elt;`**

Oops! an interrupt!

step 3: **`new_elt->next = old_head;`**

Examples of Shared Resources:

Example #1: a shared linked list

- The problem?
  - linked list structure has invariant
  - head points to first, chained through last via next field, ends with NULL
  - complete operation maintains invariant
  - partial operation does not – need atomic update

Examples of Shared Resources:

Example #2: external state

- The core problem
  - devices have state
  - processors interact with devices using specific protocols
  - protocol often requires several steps (e.g., I/O instructions)
  - device cannot differentiate which piece of code performed an operation
- Example:
  - VGA controller operations for scrolling window with color modulation
  - interrupt handler drives color manipulations
  - program handles scrolling using pixel shift
  - both use VGA attribute register (port 0x3C0)

Examples of Shared Resources:

Example #2: external state

- Protocol for attribute control register
  - 22 different attributes accessed via this register
  - first send index
  - then send data
  - VGA tracks whether next byte sent is index or data

- Problem: processor can't know which one is expected
- Solution: reading from port 0x3DA forces VGA to expect index next

Examples of Shared Resources:

Example #2: external state

- Consider the program code
  - the horizontal pixel panning register is register 0x13
  - assume that the code should write the value 0x03 to it

```
(discard) ← P[0x3DA]    MOVW $0x3DA, %DX
                        INB  (%DX), %AL

0x13 → P[0x3C0]         MOVW $0x3C0, %DX
                        MOVB $0x13, %AL
                        OUTB %AL, (%DX)

0x03 → P[0x3C0]         MOVB $0x03, %AL
                        OUTB %AL, (%DX)
```

Examples of Shared Resources:

Example #2: external state

- What happens if the interrupt occurs after the first write to 0x3C0?
  - the interrupt handler is executing basically the same code
  - leaves the VGA expecting an index

- What is the solution?

Examples of Shared Resources:

Example #3: handshake synchronization

- A device generates an interrupt after it finishes executing a command
- Consider the following attempt to synchronize

the shared variable…

```
int device_is_busy = 0;
```

the interrupt handler…

```
device_is_busy = 0;
```

Examples of Shared Resources:

Example #3: handshake synchronization

The program function used to send a command to the device…

```
while (device_is_busy); /* wait until device is free */
device_is_busy = 1;
/* send new command to device */
```

- Q: Does the loop work?
- No.
  - Compiler assumes sequential program.
  - Variables can't change without code that changes them.

Examples of Shared Resources:

Example #3: handshake synchronization

```
LOOP: MOVL device_is_busy, %EAX
      CMPL $0x0, %EAX
      JNE  LOOP
```

- Nothing can change variable, so no need to reload (move LOOP down a line).

Examples of Shared Resources:

Example #3: handshake synchronization

```
      MOVL device_is_busy, %EAX
LOOP: CMPL $0x0, %EAX
      JNE  LOOP
```

- Now nothing can change EAX, so move it down another line (to branch!).

Examples of Shared Resources:

Example #3: handshake synchronization

```
        MOVL device_is_busy, %EAX
        CMPL $0x0, %EAX
LOOP:   JNE  LOOP
```

- Will interrupt handler break you out of the resulting infinite loop?

Examples of Shared Resources:
Example #3: handshake synchronization

- Solution
  - mark variable as volatile
  - tells compiler to never assume that it hasn't changed between uses

the shared variable...

```
volatile int device_is_busy = 0;
```

- Why not mark everything volatile?
  - forces compiler to always re-load variables
  - more memory operations = slower program

Examples of Shared Resources:
Example #3: handshake synchronization

- Is it ok to swap setting the variable and sending the command?
- No.
  - introduces a race condition:

```
/* send new command to device */
---- INTERRUPT OCCURS HERE ----
device_is_busy = 1;
```

    - Next command call blocks (forever) for device to be free.

Critical Sections

- Some parts of program need to appear to execute atomically, i.e., without interruption
- Full version: atomic with respect to code in interrupt handler
  - for now, the clause is implied i.e., only interrupt handlers can operate during our programs
  - however, multiprocessors may have >1 program executing at same time

Critical Sections

- Solution?
  - IF (the interrupt enable flag)

critical section start (CLI)

(the code to be executed atomically)

critical section end (STI)

- What else must be prevented?
  - no moving memory ops into or out of critical section!

Critical Sections in Examples

- Example #2: external state

```
MOVW  $0x3DA, %DX
INB   (%DX), %AL

MOVW  $0x3C0, %DX
MOVB  $0x13, %AL

OUTB  %AL, (%DX)
MOVB  $0x03, %AL
OUTB  %AL, (%DX)
```

CLI

the critical section should be as short as possible

STI

Critical Sections in Examples

- Why should critical sections be short?
  - avoid delaying device service by interrupt handler
  - long delays can even crash system (e.g., swap disk driver timeout)

- Example #1: a shared linked list

CLI

```
old_head = head;
head = new_elt;
new_elt->next = old_head;
```

could skip first statement, but including is safer

STI

- If interrupt handler can change list, too, leaving out first inst. creates race
- Example #3: handshake synchronization—volatile suffices for this example