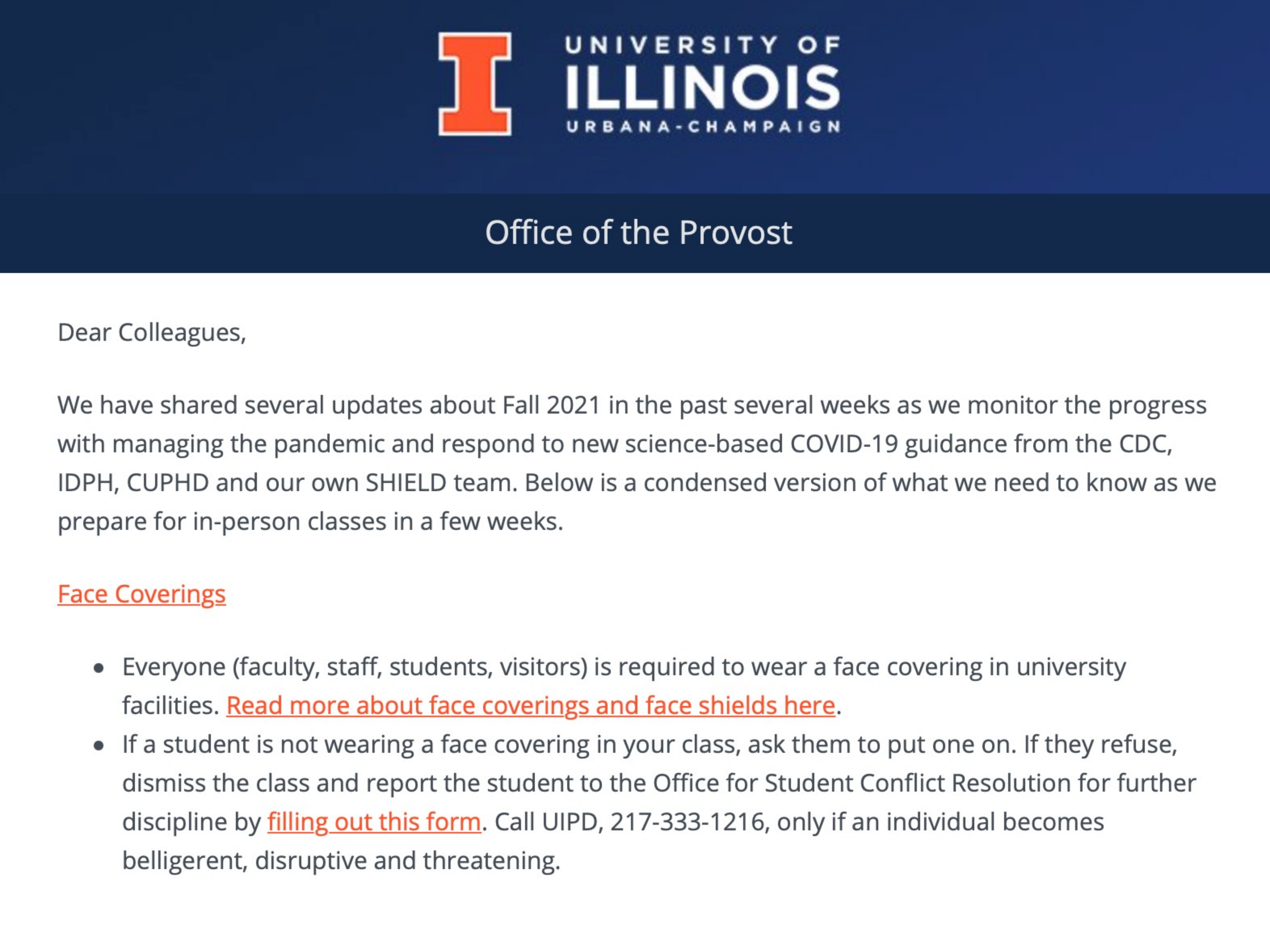
ECE391- Computer System Engineering

Lecture 10

Linux abstraction of PIC

University of Illinois at Urbana- Champaign
Fall 2021



UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Office of the Provost

Dear Colleagues,

We have shared several updates about Fall 2021 in the past several weeks as we monitor the progress with managing the pandemic and respond to new science-based COVID-19 guidance from the CDC, IDPH, CUPHD and our own SHIELD team. Below is a condensed version of what we need to know as we prepare for in-person classes in a few weeks.

Face Coverings

- Everyone (faculty, staff, students, visitors) is required to wear a face covering in university facilities. Read more about face coverings and face shields here.
- If a student is not wearing a face covering in your class, ask them to put one on. If they refuse, dismiss the class and report the student to the Office for Student Conflict Resolution for further discipline by filling out this form. Call UIPD, 217-333-1216, only if an individual becomes belligerent, disruptive and threatening.

Announcements

- PS2 Posted - Committed to the master (main) branch on GitLab by 5:59PM on 9/21
- MP2 Posted - All checkpoints should be committed to the master(main) branch on GitLab by:
  - Checkpoint 1: 5:59PM on 10/5
  - Final Checkpoint: 5:59PM on 10/12

ECE391 EXAM 1

- EXAM I – Wednesday, September 29th, 7:00pm-9:00 PM
  - Location: ECEB 1002
- NO Lecture on Tuesday, September 28
  - Review Session

Lecture Topics

- Linux abstraction of PIC
- General interrupt abstractions
- Linux interrupt system
  - data structures
  - handler installation & removal
  - invocation
  - execution
  - tasklets

Linux Abstraction of PICs

- Uses a jump table
  - same as vector table (array of function pointers)

- Table is hw_irq_controller structure (or struct irq_chip)
  - each vector # associated with a table
  - table used to interact with appropriate PIC (e.g., 8259A, or Advanced PIC)

human-readable name
startup function
shutdown function
enable function
disable function
mask function
mask_ack function
unmask function
(+ several others…)

Linux Abstraction of PICs

- *hw_irq_controller* structure definition
  - IRQs are #’d 0-15 (correspond to vector # - 0x20)

```
const char* name;
unsigned int (*startup)(unsigned int irq);
void (*shutdown)(unsigned int irq);
void (*enable)...
void (*disable)...
void (*ack)...
void (*end)...
/* we’ll ignore the others... */
```

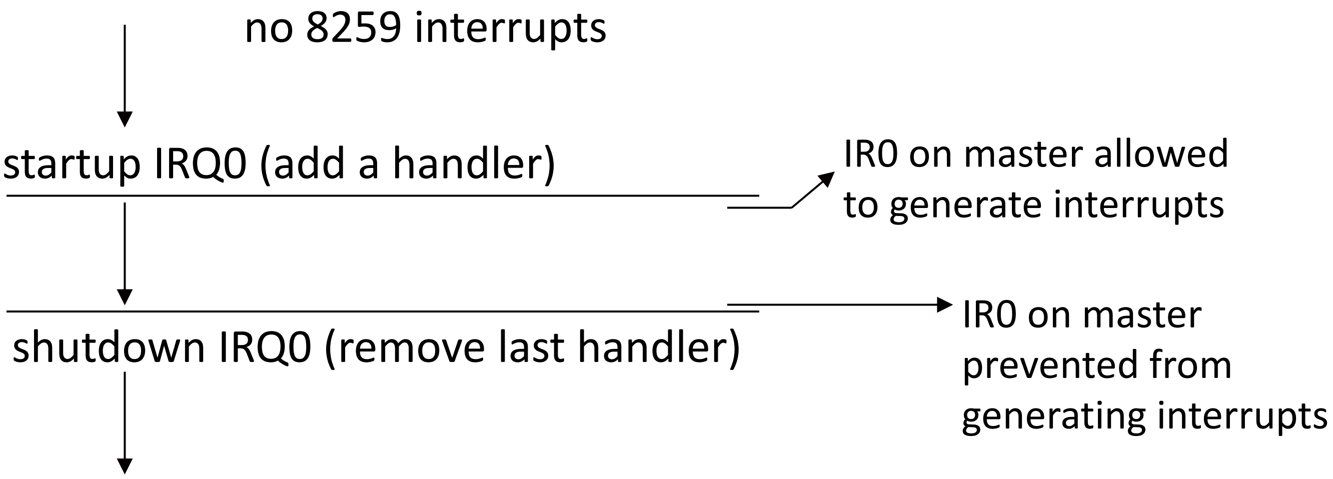
8259A’s human-readable name is “XT-PIC”; see /proc/interrupts

PIC Functions in Jump Table: Explanation

- Initially, all 8259A interrupts are masked out using mask on 8259A
- startup and shutdown functions
  - startup is called when first handler is installed for an interrupt
  - shutdown is called after last handler is removed for an interrupt
  - both functions change the corresponding mask bit in 8259A implementation

PIC Functions in Jump Table: Explanation (cont.)

Example



no 8259 interrupts

startup IRQ0 (add a handler)

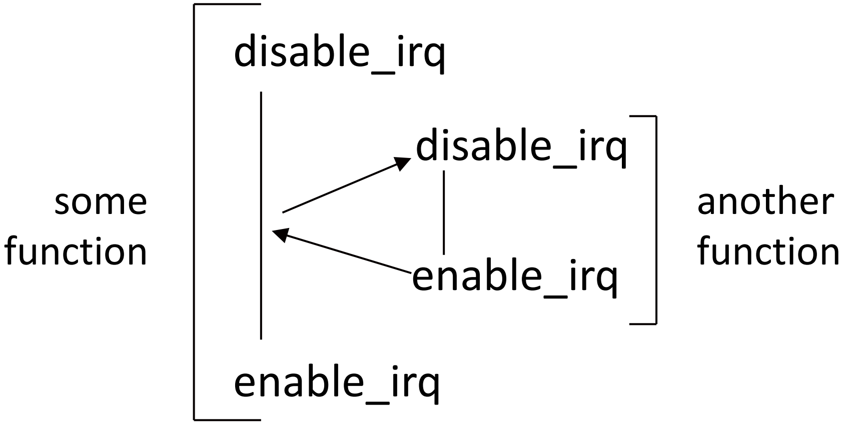
IR0 on master allowed to generate interrupts

shutdown IRQ0 (remove last handler)

IR0 on master prevented from generating interrupts

PIC Functions in Jump Table (cont.)

- disable/enable functions
  - used to support nestable interrupt masking (disable_irq, enable_irq)
  - on 8259
    - first disable_irq calls jump table disable, which masks interrupt on PIC
    - last enable_irq calls jump table enable, which unmasks interrupt on PIC



some function

disable_irq

disable_irq

enable_irq

another function

enable_irq

disable/enable pairs must match!

PIC Functions in Jump Table (cont.)

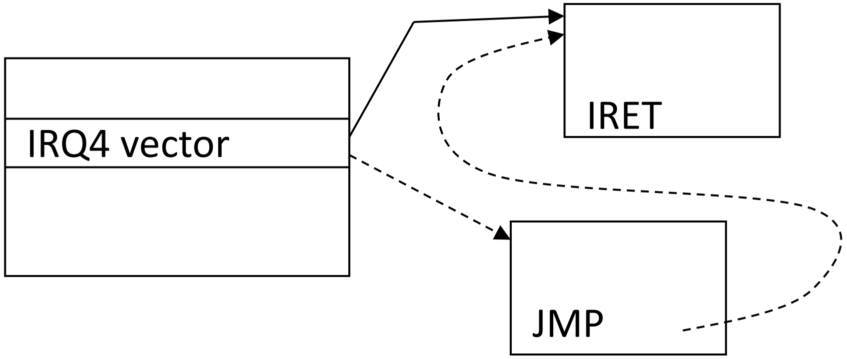
- mask_ack function
  - called at start of interrupt handling to ack receipt of the interrupt
  - on 8259 (mask and ack), masks interrupt on PIC, then sends EOI to PIC
- umask function
  - called at end of interrupt handling
  - on 8259, enables interrupt (unmasks it) on PIC

General Interrupt Abstractions: Interrupt Chaining

- Hardware view: 1 interrupt → 1 handler
- Problems
  - may have > 15 devices
  - > 1 software routines may want to act in response to device
  - examples:
    - hotkeys for various functions
    - move mouse to lower-right corner to start screen-saver

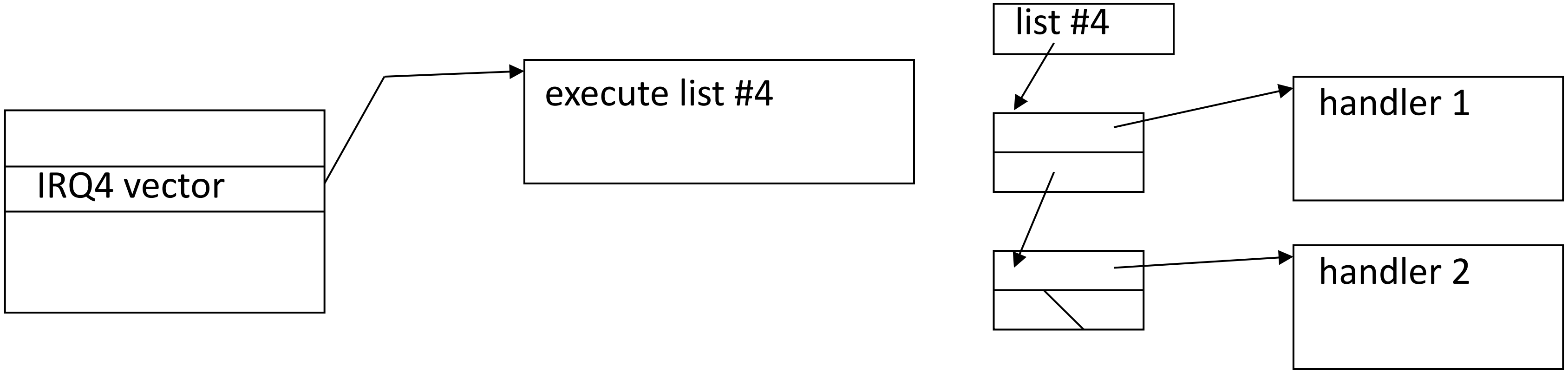
General Interrupt Abstractions: Interrupt Chaining (cont.)

- One approach
  - used by terminate and stay resident (TSR) programs in DOS
  - form linked list (chain) of handlers using JMP instructions
  - not very clean
    - no way to remove self
    - unless you're first in list
  - to be fair
    - TSR program not designed for removal
    - but also can't restart, etc.



General Interrupt Abstractions Interrupt Chaining (cont.)

- Solution
  - interrupt chaining with linked list data structure
  - (not list embedded into code!)



General Interrupt Abstractions: Interrupt Chaining (cont.)

- Drawbacks of chaining
    - for > 1 device
        - must query devices to see if they raised interrupt
        - not always possible
    - for 1 device
        - must avoid stealing data/confusing device
        - example
            - by sending two characters to serial port
            - in response to interrupt declaring port ready for one char.
        - another example
            - reading mouse location twice
            - if device protocol specifies reading once per interrupt
- Bottom line
    - Effectively impossible to make two pieces of code work together without planning

General Interrupt Abstractions: Soft Interrupts (cont.)

- Recall: why support interrupts?
  - slow device gets timely attention from fast processor
  - processor gets device responses without repeatedly asking for them
- A useful concept in software
  - example: network encryption/decryption
  - packet arrives, given to decrypter
  - when decrypter (software program) is done
    - want to interrupt program
    - to transfer data from packet
  - but has no access to INTR pin

General Interrupt Abstractions: Soft Interrupts (cont.)

- Solution
  - software-generated (soft) interrupt
  - (similarly, but later, signals—user-level soft interrupts)
  - runs at priority between program and hard interrupts
  - usually generated
    - by hard interrupt handlers
    - to do work not involving device
- Linux version is called tasklets
  - used by code provided to you for MP1
  - discussed later