

Interrupts and Signals

CMSC421

21 February 2018

But first

- My dog ate my homework
- Computer have moving parts and humans that write software, and stuff happens.
- With your Google drive, you have one terabyte of storage

Interrupts

- Source:
 - Angela Demke Brown, University of Toronto

Summary

- Difference between interrupts and signals
- Polling vs. interrupts
 - Polling might be better for real time device
- Hardware interrupt handling and software interrupt handling

Interrupts

- An interrupt is an event external to the currently executing process that causes a change in the normal flow of instruction execution. This is usually generated by hardware devices.
 - From “Design and Implementation of the FreeBSD Operating System”, Glossary
- Key point: interrupts are asynchronous with respect to the current process. This means the process cannot determine when the next interrupt will happen.
 - This could be a disk drive has some data ready for your process
 - Perhaps a network has some data
- Signals are generated by processes running in the computer.

Why do we have interrupts?

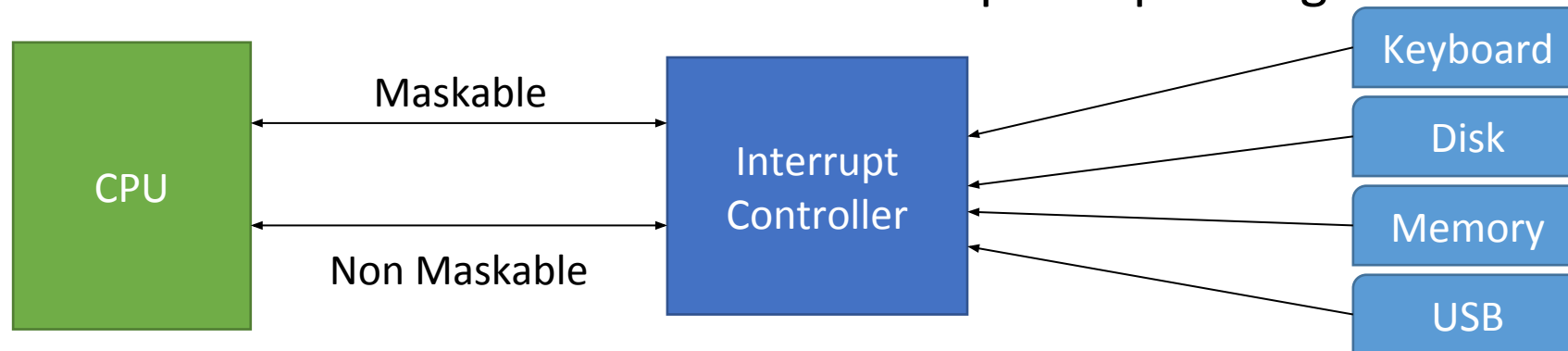
- We have so many external devices connected to our computer
 - How many different ways can a user change the operation of an iPhone?
 - For example, the power button, Class exercise, ...
 - Keyboard, mouse, screen, diskdrive
 - Camera
- These devices occasionally need CPU service
 - However, we cannot predict when.
 - External events typically happen infrequently
- But, we want the expensive CPU to stay busy.

Possible Solution: Polling

- CPU periodically checks each device to see if it needs service
- Issues
 - It could take the CPU time to ask **each** device if it has anything ready
 - It ask fewer times, but the response time will be reduced
 - This could be efficient if events arrive rapidly because the CPU is always polling
- “Polling is like picking up your phone every few seconds to see if you have a call...”

Interrupts

- Give each device a wire that it can use to signal the processor
 - When the interrupt signaled, processor executes an interrupt handler to deal with the interrupt.
 - The keyboard handler is going to be different than the memory handler
 - Benefit: No overhead when there are no requests pending

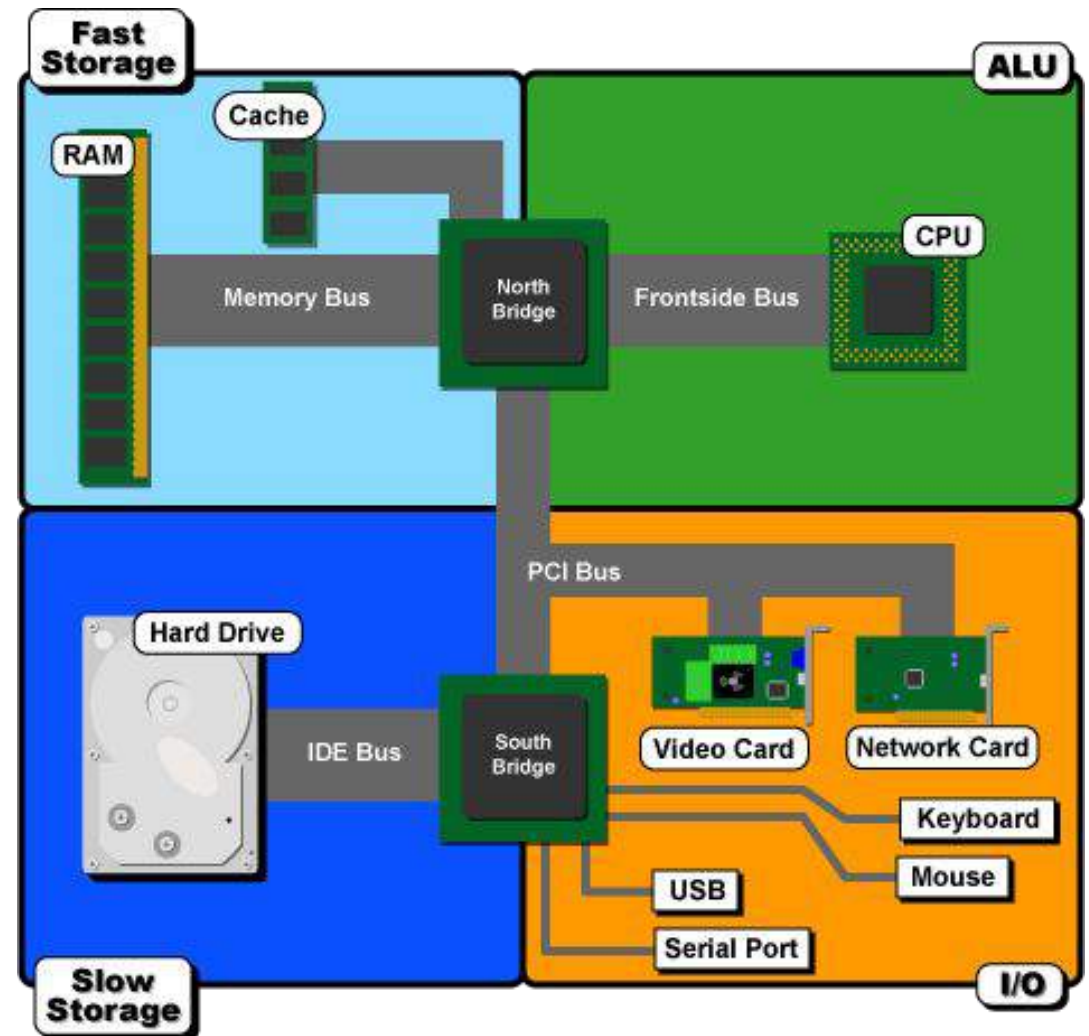


Intel 430HX Motherboard

- Programmable interrupt controller (PIC) part of the Southbridge chip
- 8 inputs, 1 output
- Top to bottom, fast to slow

Image:

<http://origin.arstechnica.com/articles/paedia/hardware/mobo-guide-1.media/430HX.png>



Polling vs. Interrupts, which is better?

- **Polling:** picking up your phone every few seconds, is anyone there?
- **Interrupts:** waiting for the phone to ring
- **Interrupts** the system as a whole is faster if the processor is doing other things and response time is not critical
- **Polling** is better if the processor has to respond to a device immediately.
 - This sounds good for real time systems
 - The CPU must respond to this event in a specified time.
 - Could have a secondary controller to handle this

Hardware Interrupt Handling

- Interrupt controller signals the CPU that the interrupt has occurred and passes the interrupt number
 - Interrupts are assigned priorities to handle simultaneous interrupts
 - Lower priority interrupts may be disabled during service
- CPU checks interrupt request after every instruction: if raised, then:
 - Uses interrupt number to determine which handler to start
 - Interrupt vector associates handlers with interrupts
- Basic program state saved
- CPU jumps to interrupt handler
- When an interrupt service routine is complete, the original program state (where it left off) is reloaded and program continues.

Software Interrupt Handling

- Typically, two parts to interrupt handling
 - The part that has to be done now
 - So the device can continue working
 - The part that can be saved for later
 - To make the device respond faster
 - And, have a more convenient execution context
 - What?

Interrupt Context

- Execution of the first part of the interrupt handler uses the context of whatever was interrupted
 - The process state is saved in process structure
 - Handler uses the interrupted thread's kernel stack
 - Careful about stack-allocated data
 - Handler is not allowed to block
 - Has no process structure of its own to save state or allow rescheduling
 - Cannot call functions that might block
- Handler must be kept fast and simple
 - Typically, set up for the second part, fast that second part needs to execute and re-enables interrupt.

Software Interrupts

- The deferred part of interrupt handling is sometimes called “software interrupts”
 - Linux calls them the “bottom halves”
- What things can be deferred?
 - Networking
 - Time critical work is 1. Copy packet off hardware 2. respond to hardware
 - Deferred work 1. process packet 2. Pass packet to the correct application
 - Timers
 - Time critical work is 1. Increment current time of day
 - Deferred – recalculate process priorities

Free BSD 5.2 and up

- All h/w devices and other interrupt events have an associated kernel thread with a suitable priority
- First part of the interrupt handling just schedules the proper thread to run
 - Interrupted thread is marked as needing reschedule
 - High-priority handler thread is the scheduled on “return from interrupt”
- Handlers have full context and a separate stack
 - So they can block
- Still common to have two parts with the second part having a lower priority
- Some interrupts that have to be fast run entirely in interrupt context (clock)

Signals

- Software equivalent of hardware interrupts
- Allows processes to respond to asynchronous external events
 - Process may specify its own signal handlers or use OS default
 - Defaults
 - Ignore signal
 - Terminate all threads in process
 - Stop all threads in process
 - Resume all threads in process
- Used as simple Inter-process communication.

Signal Basics

- Process structure has flags for possible signals and actions to take
- When the signal is posted to process, signal pending flag is marked
- When process is next scheduled to run, pending signals are checked and action taken
 - Signal delivery is not instantaneous

Summary

- Difference between interrupts and signals
- Polling vs. interrupts
 - Polling might be better for real time device
- Hardware interrupt handling and software interrupt handling

Last slide

- See subject