CS 3502
Operating Systems

Interrupt

Kun Suo

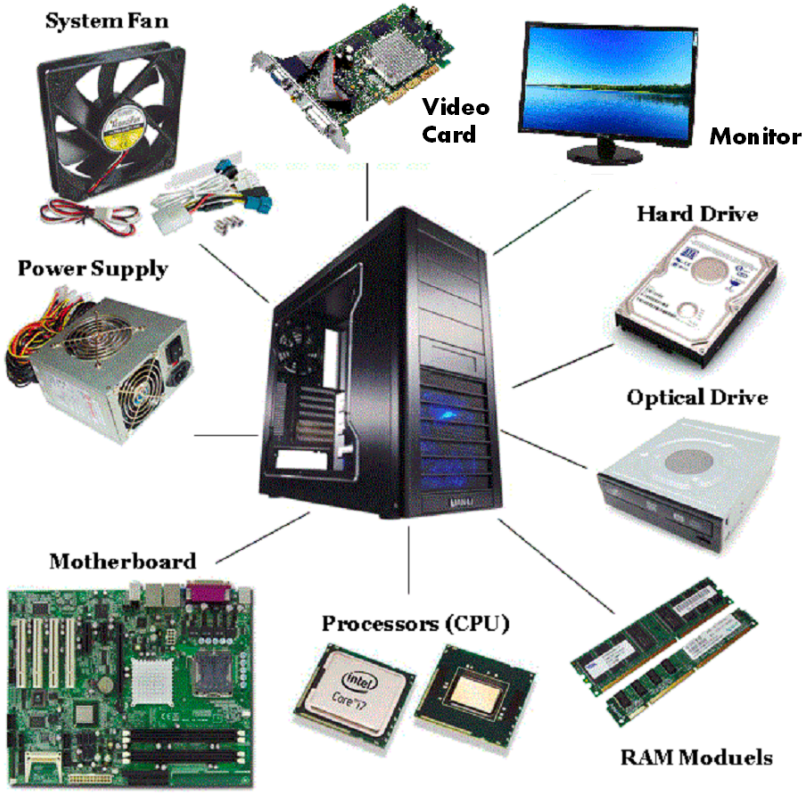
Computer Science, Kennesaw State University

https://kevinsuo.github.io/

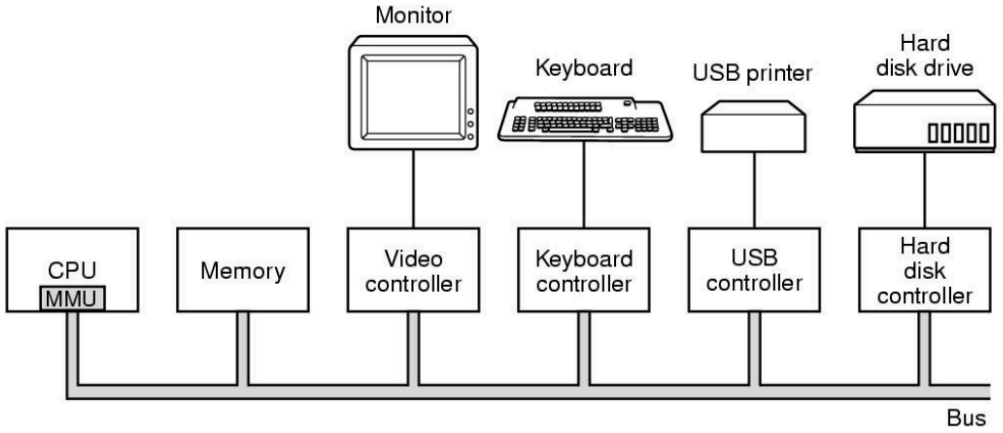
Outline

- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity

How can the processor work with hardware



When event happens
When task finishes
When job is ready
When user inputs something
When something is broken
…



What is Polling?

- How can the processor work with hardware without impacting the machine's overall performance?
  - **Polling: repeatedly check whether the hardware is ready or not**

Polling

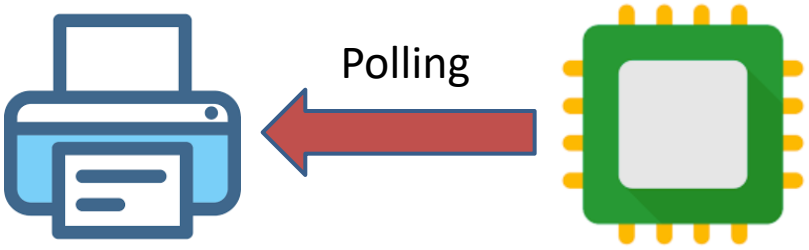
Polling Drawbacks

- How can the processor work with hardware without impacting the machine's overall performance?
  - **Polling: repeatedly check whether the hardware is ready or not**

Randomly Every Ns/min

Polling: overhead

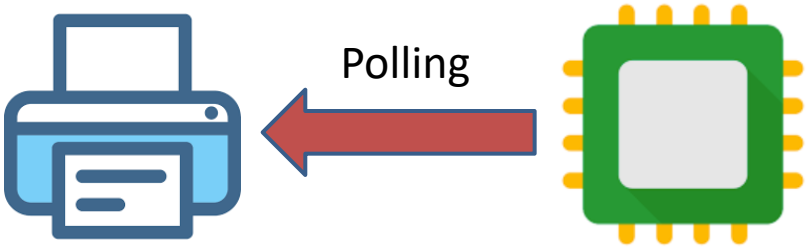
1, Increase system overhead. Whether it is task polling or timer polling, it needs to consume the corresponding system resources.



What is Polling?

- How can the processor work with hardware without impacting the machine's overall performance?
  - **Polling: repeatedly check whether the hardware is ready or not**

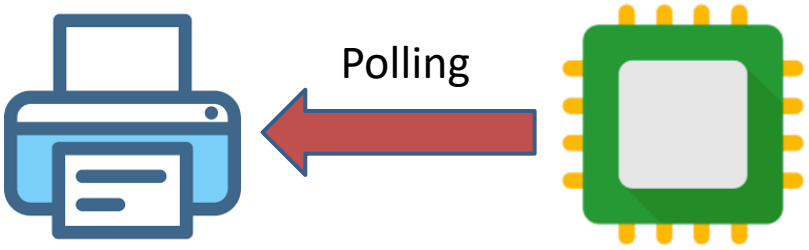
Polling: (timely)
2, Unable to sense device status changes in a timely manner. Device state changes during the polling interval can only be discovered on the next poll, which will not be able to meet real-time sensitive applications.



What is Polling?

- How can the processor work with hardware without impacting the machine's overall performance?
  - **Polling: repeatedly check whether the hardware is ready or not**

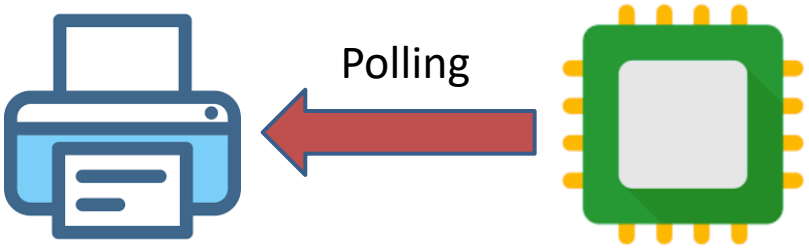
Polling: resource efficiency
3, waste CPU resources. Polling is always in progress regardless of whether the device has changed state. In the real world, the state change of most devices is usually not so frequent, and polling idle will waste CPU time slices.



What is Polling?

- How can the processor work with hardware without impacting the machine's overall performance?
  - **Polling: repeatedly check whether the hardware is ready or not**

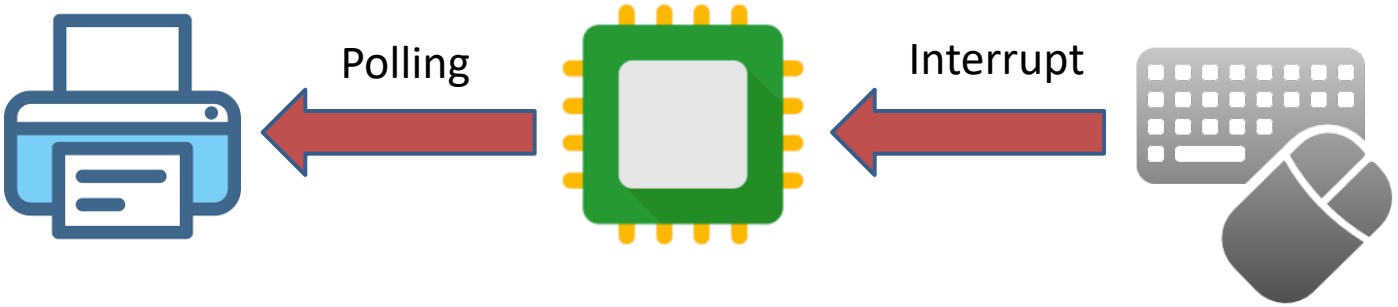
Polling:
1, increase system overhead
2, cannot detect in time
3, waste of CPU resources



What is interrupt?

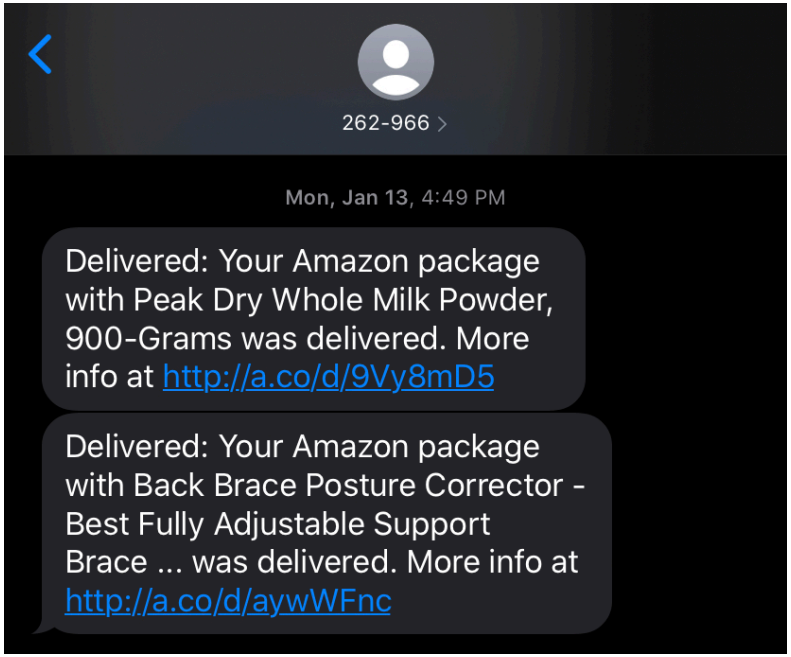
- How can the processor work with hardware without impacting the machine's overall performance?
  - Polling: repeatedly check whether the hardware is ready or not
  - **Interrupt: hardware signals to the kernel when attention is needed**

Polling:
1, increase system overhead
2, cannot detect in time
3, waste of CPU resources

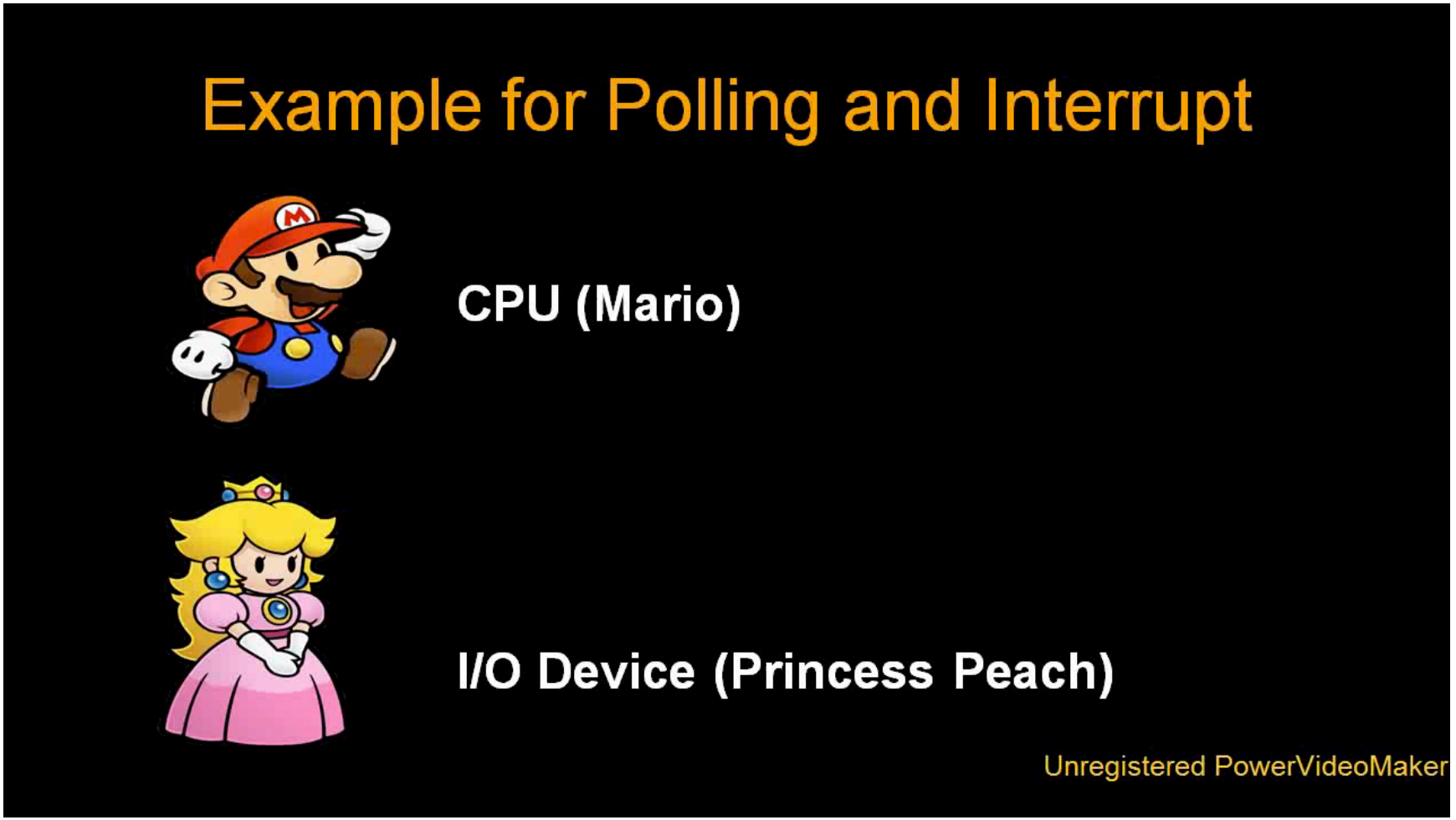


Interrupt


AT&T
8:32 PM
Reid
02:09
mute
keypad
speaker
add call
FaceTime
contacts


262-966
Mon, Jan 13, 4:49 PM
Delivered: Your Amazon package with Peak Dry Whole Milk Powder, 900-Grams was delivered. More info at http://a.co/d/9Vy8mD5
Delivered: Your Amazon package with Back Brace Posture Corrector - Best Fully Adjustable Support Brace ... was delivered. More info at http://a.co/d/aywWFnc

Interrupt vs. Polling



Reference: https://www.youtube.com/watch?v=M3nXI_86uLE

Interrupt vs. Polling

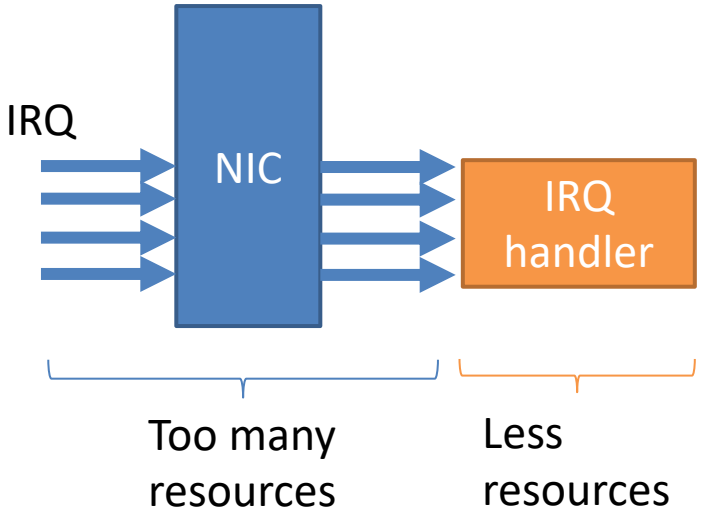
- Polling:
  - increase system overhead
  - cannot detect in time
  - waste of CPU resources

- Interrupt
  - lightweight and save CPU resources
  - handle in time

What will happen if you have too many phone calls?

1. do not have time to work on valuable things
2. might miss some important calls and tasks
3. loss data

Interrupt problem



Interrupt:

Too many **interrupt**

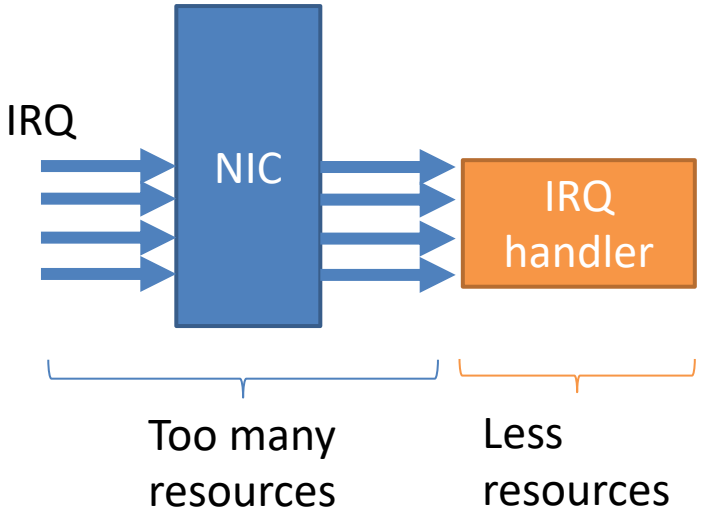
-> Less resources for **IRQ handler**

-> miss some interrupt and data

Phone sounds

Answer the call

Combine Interrupt & Polling

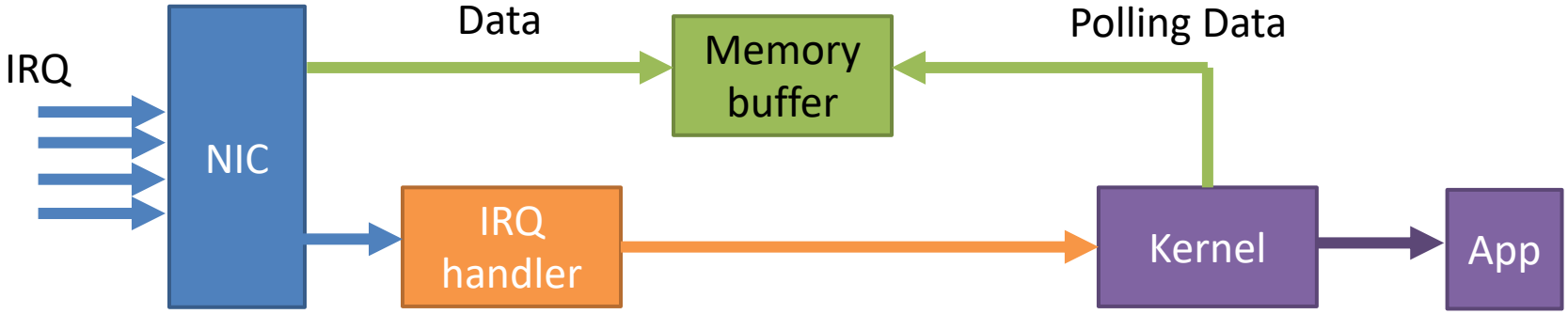


Interrupt:

Too many **interrupt**

-> Less resources for **IRQ handler**

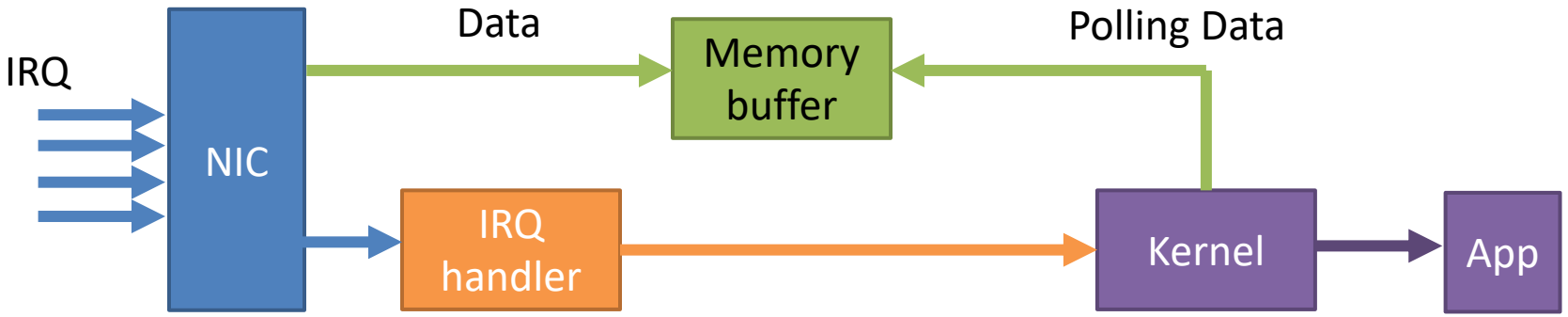
-> miss some interrupt and data



NAPI: network receiving

Combine Interrupt & Polling

- Benefits
  - Avoid frequent interruptions when receiving large amounts of data or requests
  - No need to spend a lot of CPU resources for polling



NAPI: network receiving

Interrupt: Quick look

- $ cat /proc/interrupts

- $ watch -n 1 cat /proc/interrupts

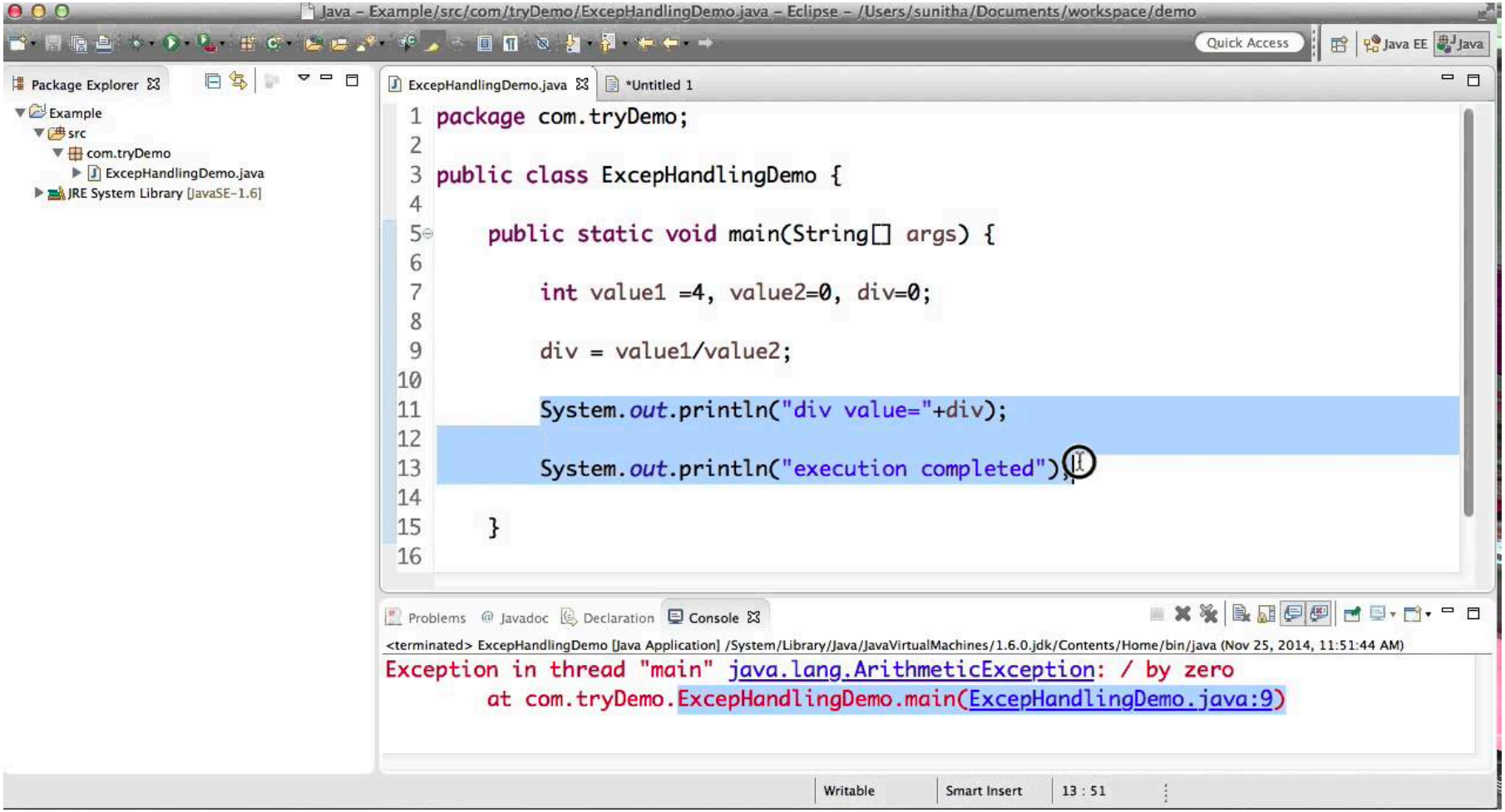
```
           CPU0       CPU1
 0:          1          0   IO-APIC   2-edge      timer
 1:          0         77   IO-APIC   1-edge      i8042
 8:          1          0   IO-APIC   8-edge      rtc0
 9:          0          0   IO-APIC   9-fasteoi   acpi
12:         15          0   IO-APIC  12-edge      i8042
14:          0          0   IO-APIC  14-edge      ata_piix
15:          0    1028751   IO-APIC  15-edge      ata_piix
16:          1        218   IO-APIC  16-fasteoi   vmwgfx
17:          0     477023   IO-APIC  17-fasteoi   ioc0
24:          0          0   PCI-MSI 344064-edge      PCIe PME, pciehp
25:          0          0   PCI-MSI 346112-edge      PCIe PME, pciehp
26:          0          0   PCI-MSI 348160-edge      PCIe PME, pciehp
27:          0          0   PCI-MSI 350208-edge      PCIe PME, pciehp
```

Exceptions

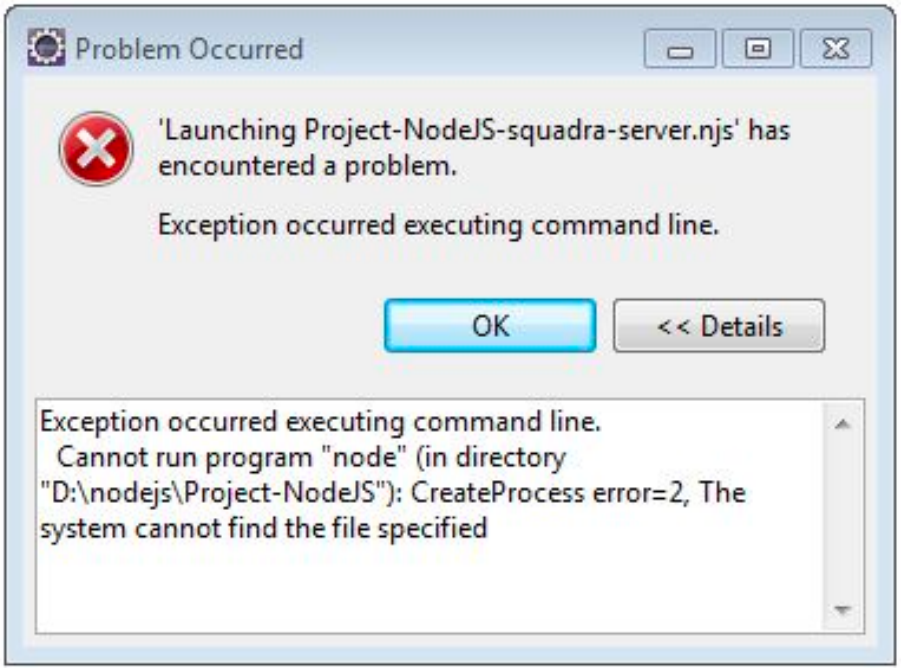
- An exception indicates that code running on the CPU has created a situation that the processor needs help to address.

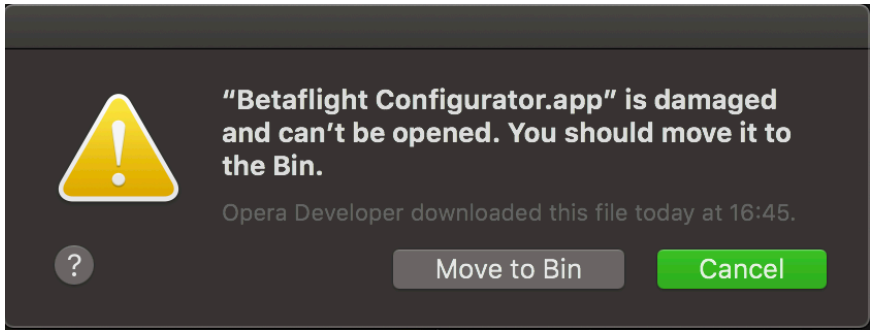
- Can you think of examples of software exceptions?
  - Divide by zero -- probably kills the process.
  - Attempt to use a privileged instruction - - also probably kills the process.
  - Attempt to use a virtual address that the CPU does not know how to translate -- a common exception handled transparently as part of virtual memory management.

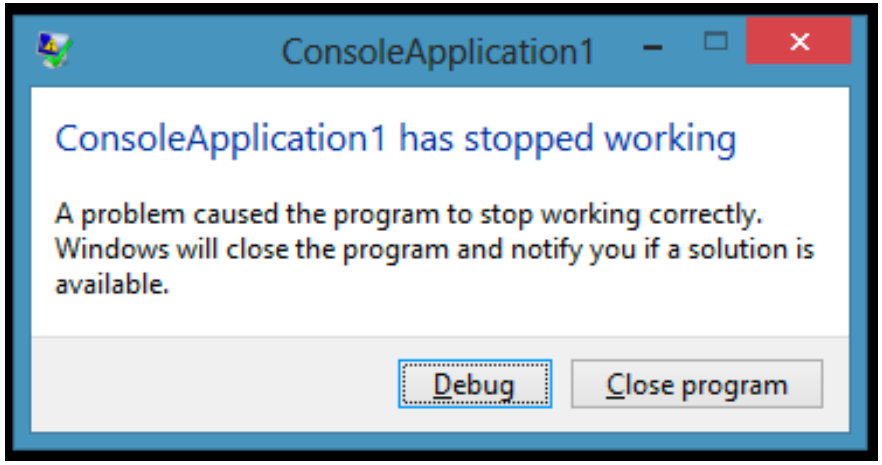
Exceptions example



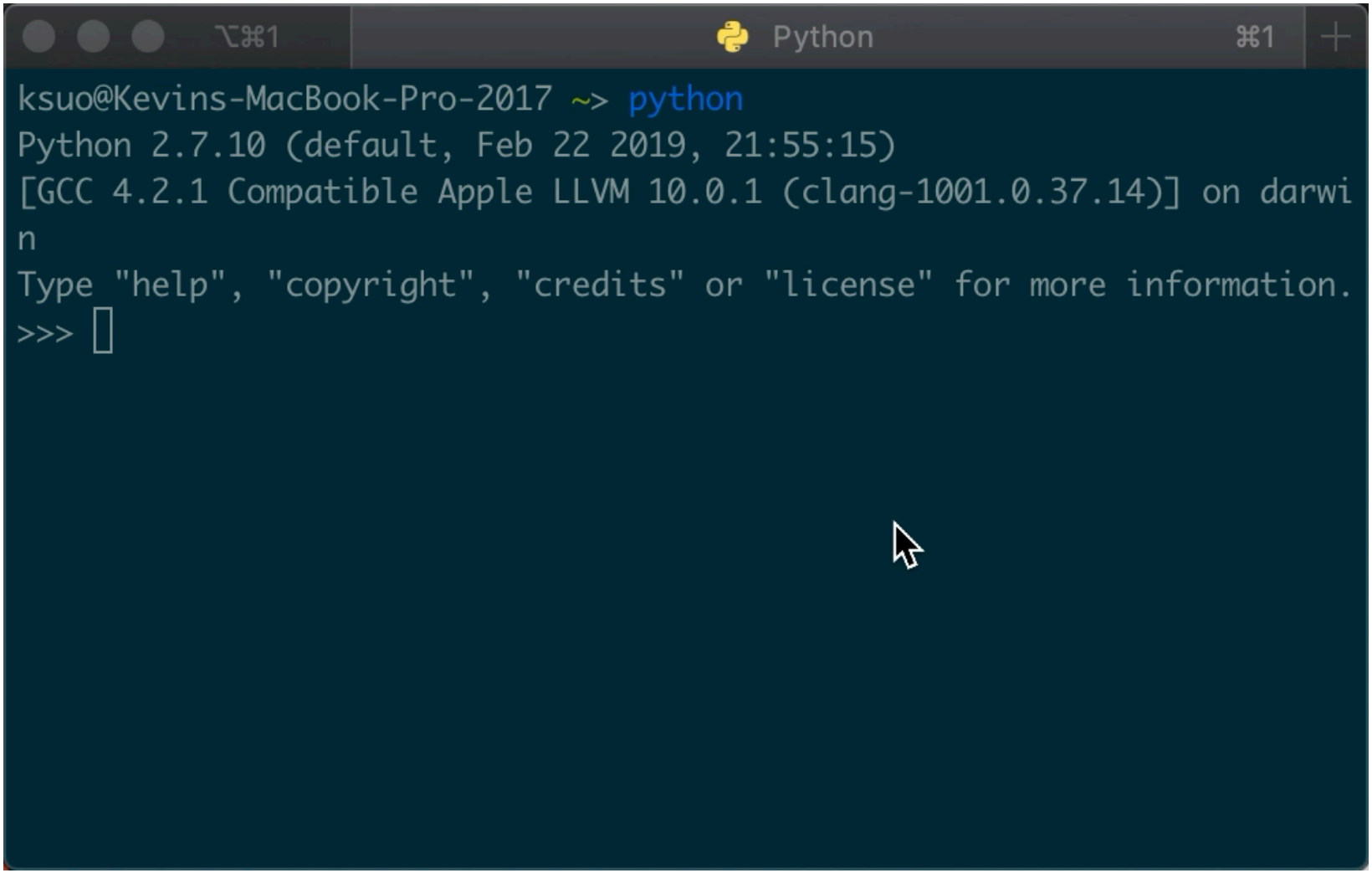
Exceptions example







Exceptions example



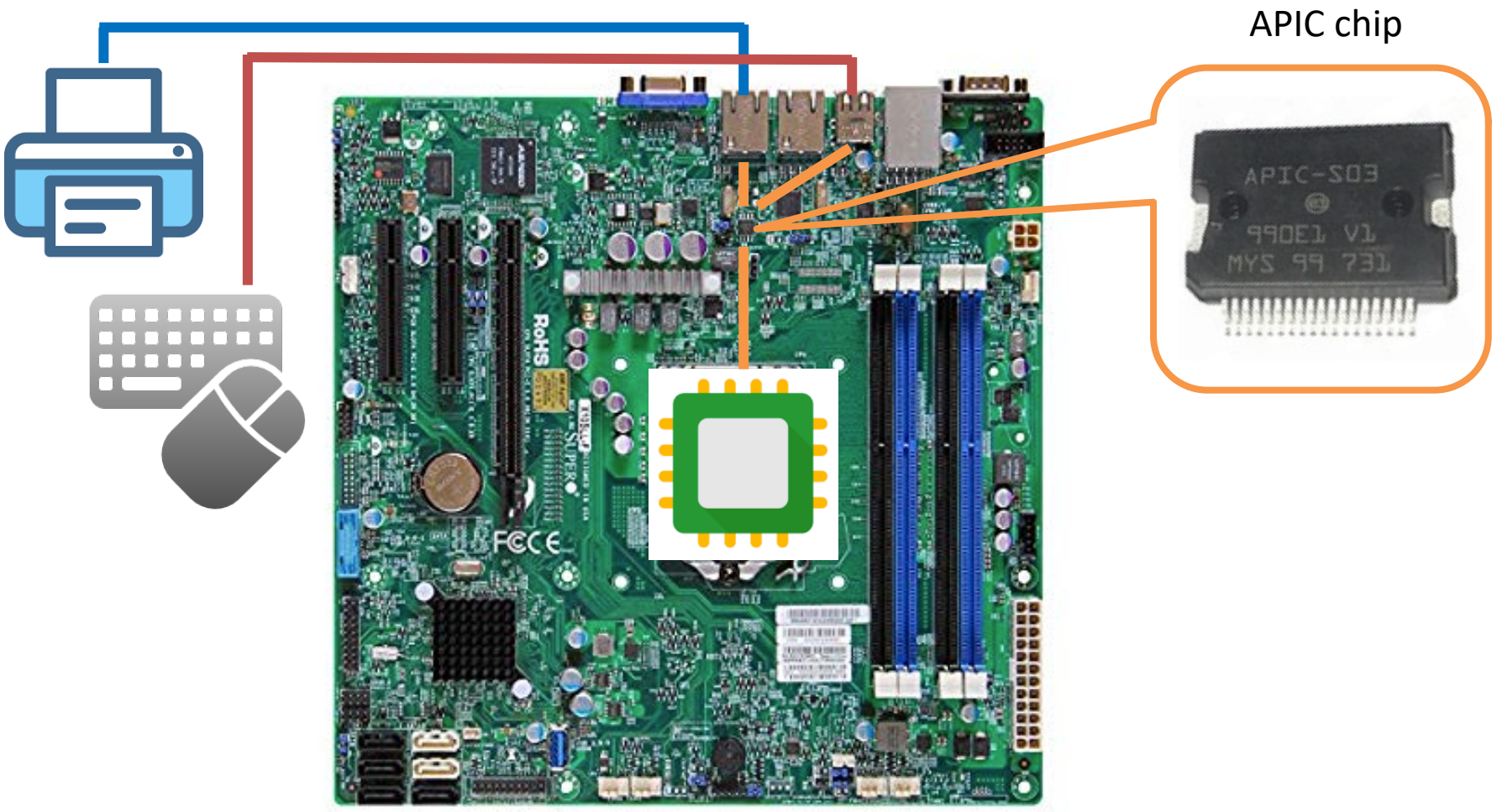
Interrupt vs. Exception

- Interrupts are **voluntary**.
  - CPU → kernel: “The process actively asks for assistance.”
- Exceptions are **non-voluntary**.
  - CPU → kernel: "It just tried to divide by zero and I think it needs to be terminated. I need some help with this process. "

Outline

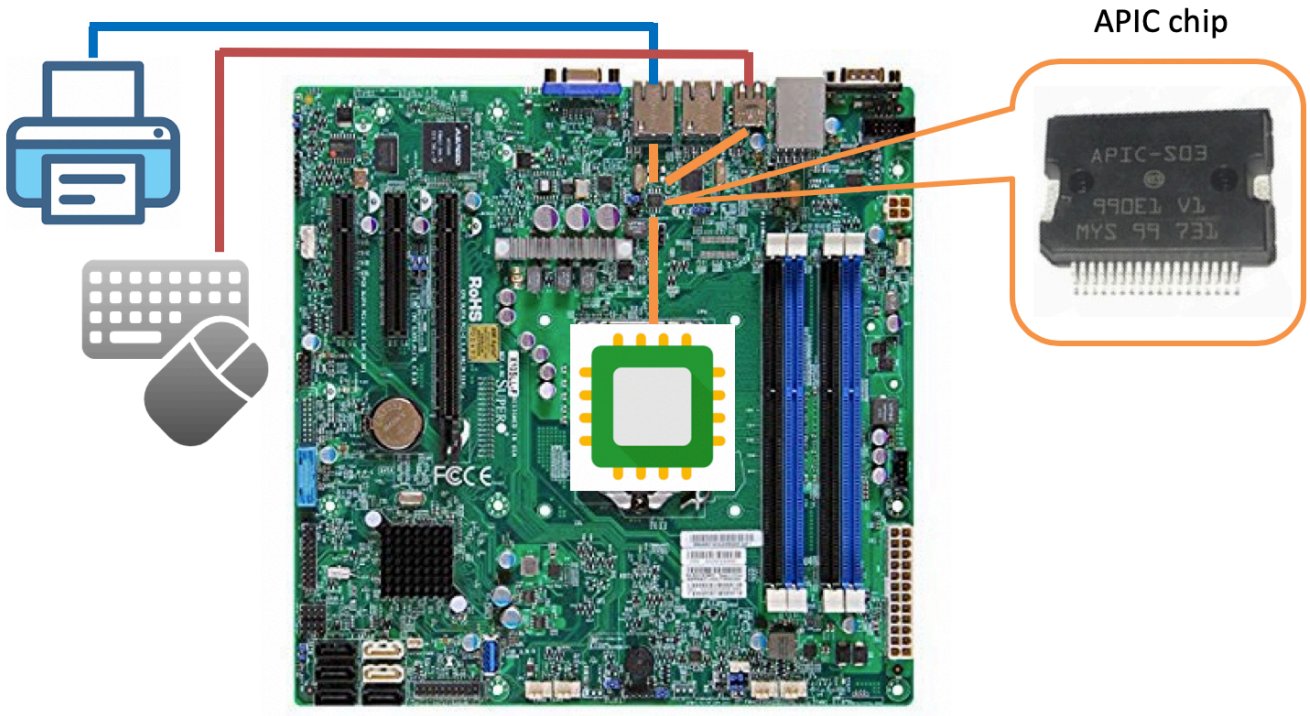
- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity

APIC: Advanced programmable interrupt controller



APIC: Advanced programmable interrupt controller

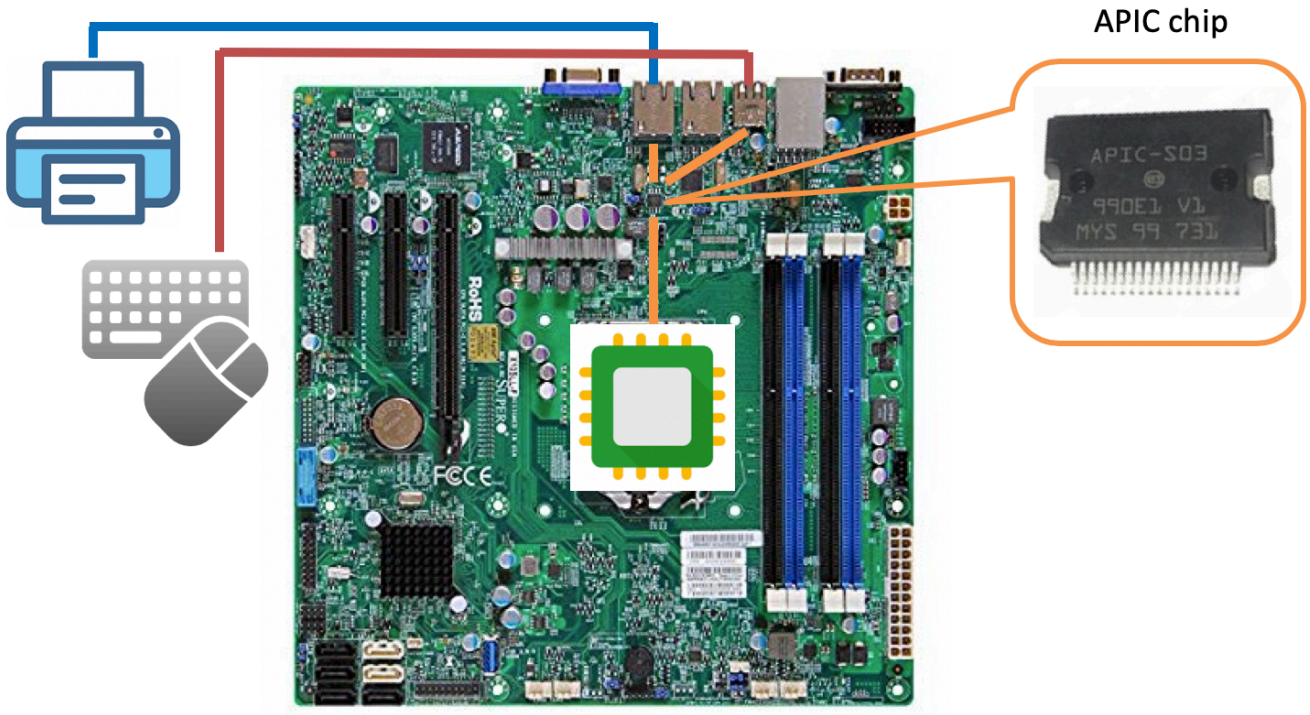
1. Responsible for telling the CPU when a specific external device wishes to ‘interrupt’
   - Needs to tell the CPU which one among several devices is the one needing service



APIC: Advanced progra

Resource	Device	Status
IRQ 0	System timer	OK
IRQ 3		OK
IRQ 4	Communications Port (COM1)	OK
IRQ 6	Standard floppy disk controller	OK
IRQ 8	System CMOS/real time clock	OK
IRQ 10	NVIDIA nForce PCI System Management	OK
IRQ 10	Multimedia Audio Controller	OK
IRQ 13	Numeric data processor	OK
IRQ 14	ATA Channel 0	OK
IRQ 15	ATA Channel 1	OK
IRQ 18	Realtek RTL8139/810x Family Fast Ethernet NIC	OK

2. APIC translates IRQ to interrupt number
   - Raises interrupt to CPU
   - Interrupt # available in register

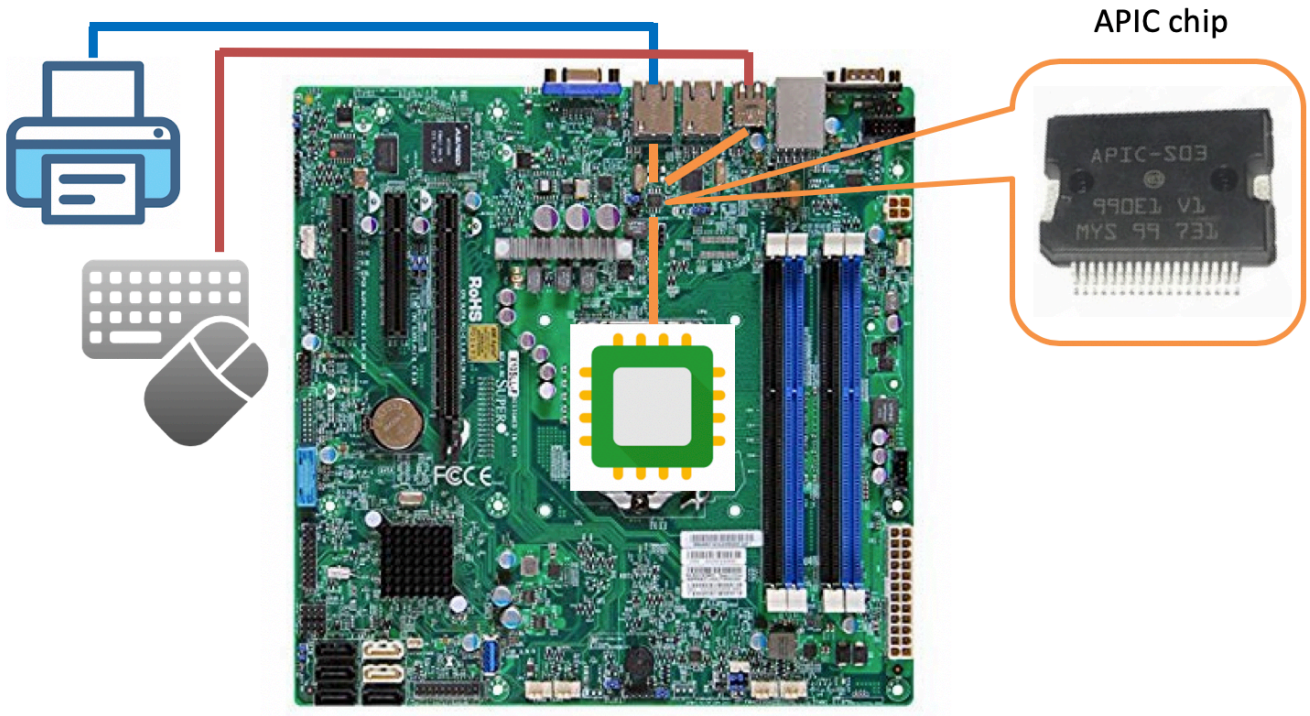


APIC: Advanced programmable interrupt controller

3. Interrupts can have varying priorities

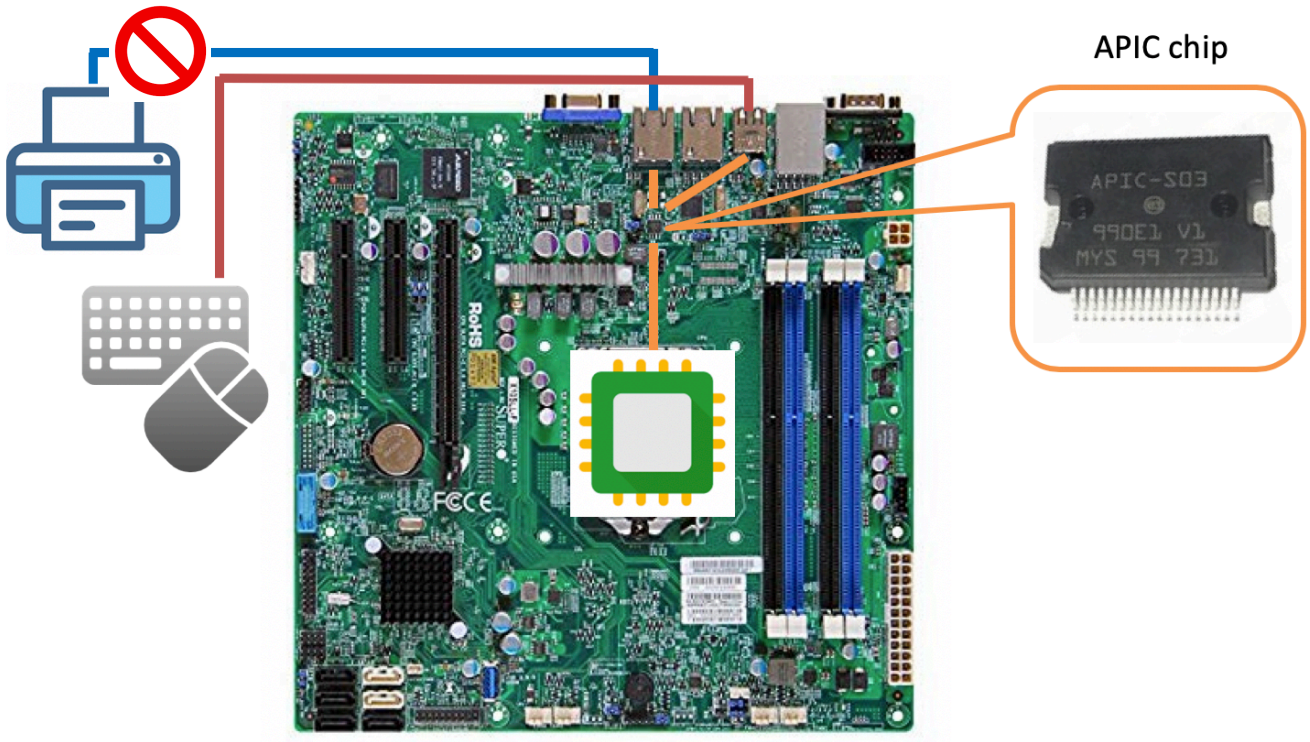
- APIC also needs to prioritize multiple requests

Mouse>Printer



APIC: Advanced programmable interrupt controller

4. Possible to “mask” (disable) interrupts at PIC or CPU



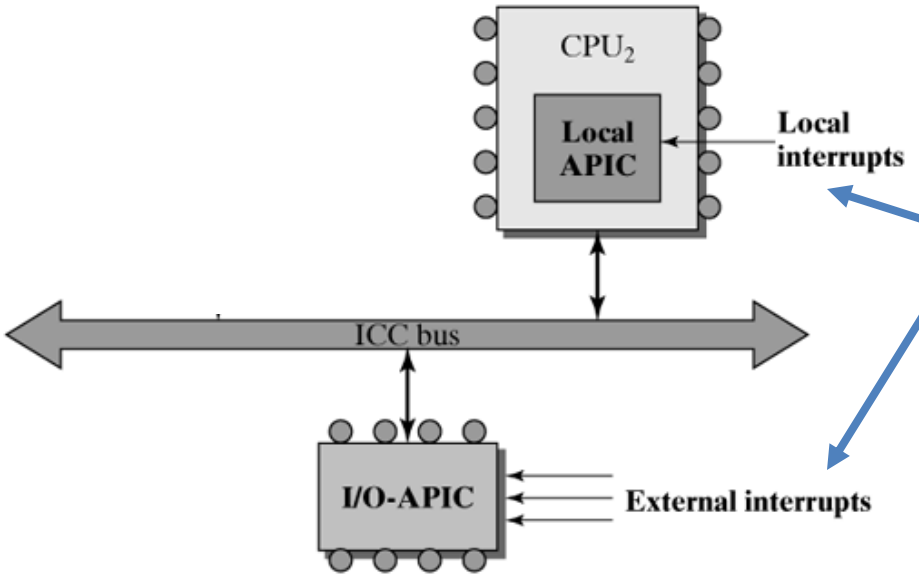
APIC: Advanced programmable interrupt controller

1. Responsible for telling the CPU when a specific external device wishes to 'interrupt'
   - Needs to tell the CPU which one among several devices is the one needing service
2. APIC translates IRQ to interrupt number
   - Raises interrupt to CPU
   - Interrupt # available in register
3. Interrupts can have varying priorities
   - APIC also needs to prioritize multiple requests
4. Possible to "mask" (disable) interrupts at PIC or CPU

APIC:
cat /proc/interrupts

VM on Amazon EC2

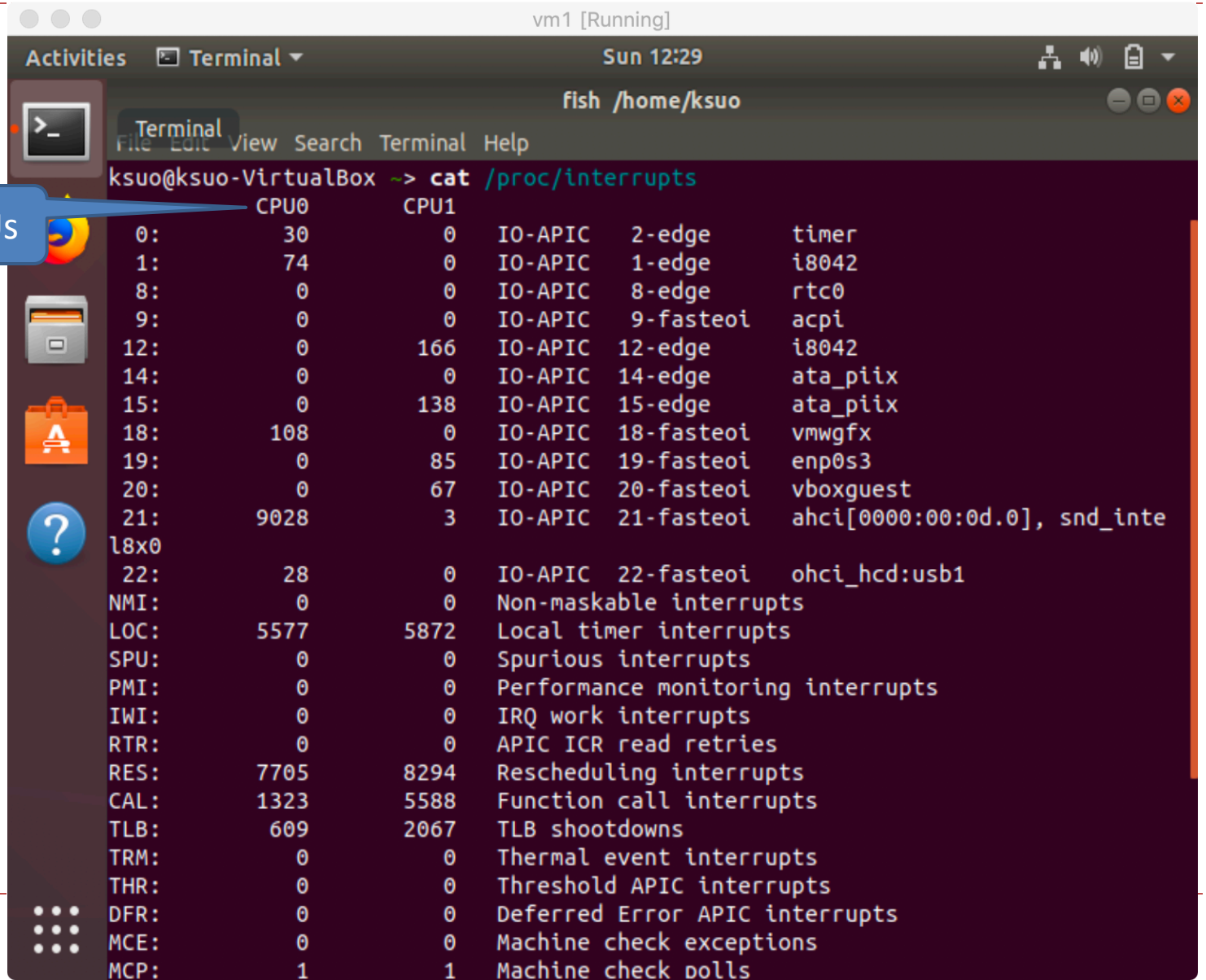
One CPU




```
1. ec2-user@ip-172-31-36-115:~ (ssh)
fish /home/pi/D... ⌘1   ec2-user@ip-172-... ⌘2
[ec2-user@ip-172-31-36-115 ~]$ cat /proc/interrupts
           CPU0
  0:         47   IO-APIC   2-edge      timer
  1:          9  xen-pirq   1-ioapic-edge  i8042
  4:       2599  xen-pirq   4-ioapic-edge  ttyS0
  8:          2  xen-pirq   8-ioapic-edge  rtc0
  9:          0  xen-pirq   9-ioapic-level  acpi
 12:         86  xen-pirq  12-ioapic-edge  i8042
 14:          0   IO-APIC  14-edge      ata_piix
 15:          0   IO-APIC  15-edge      ata_piix
 48:       4827  xen-percpu    -virq      timer0
 49:          0  xen-percpu    -ipi       resched0
 50:          0  xen-percpu    -ipi       callfunc0
 51:          0  xen-percpu    -virq      debug0
 52:          0  xen-percpu    -ipi       callfuncsingle0
 53:          0  xen-percpu    -ipi       spinlock0
 54:        254   xen-dyn    -event     xenbus
 55:      19103   xen-dyn    -event     eth0
 56:       6784   xen-dyn    -event     blkif
NMI:          0   Non-maskable interrupts
LOC:          0   Local timer interrupts
SPU:          0   Spurious interrupts
PMI:          0   Performance monitoring interrupts
IWI:          0   IRQ work interrupts
RTR:          0   APIC ICR read retries
RES:          0   Rescheduling interrupts
CAL:          0   Function call interrupts
TLB:          0   TLB shootdowns
TRM:          0   Thermal event interrupts
THR:          0   Threshold APIC interrupts
DFR:          0   Deferred Error APIC interrupts
MCE:          0   Machine check exceptions
MCP:          1   Machine check polls
HYP:      32046   Hypervisor callback interrupts
ERR:          0
MIS:          0
PIN:          0   Posted-interrupt notification event
NPI:          0   Nested posted-interrupt event
PIW:          0   Posted-interrupt wakeup event
```

APIC: cat /proc/interrupts

two CPUs

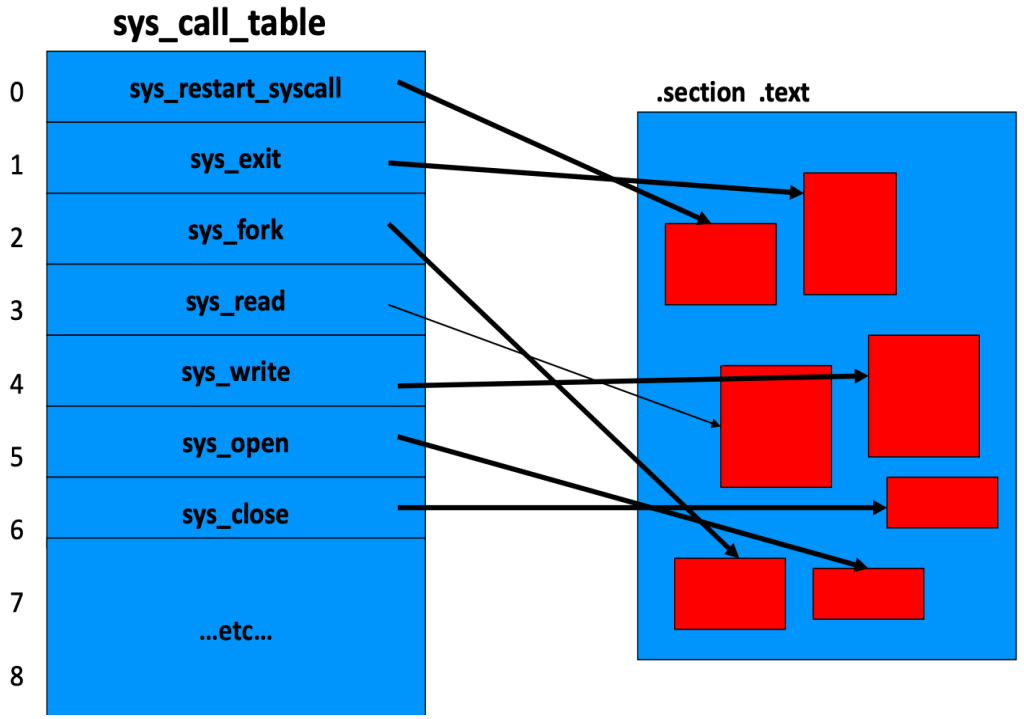


Outline

- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity

Similar as System call table

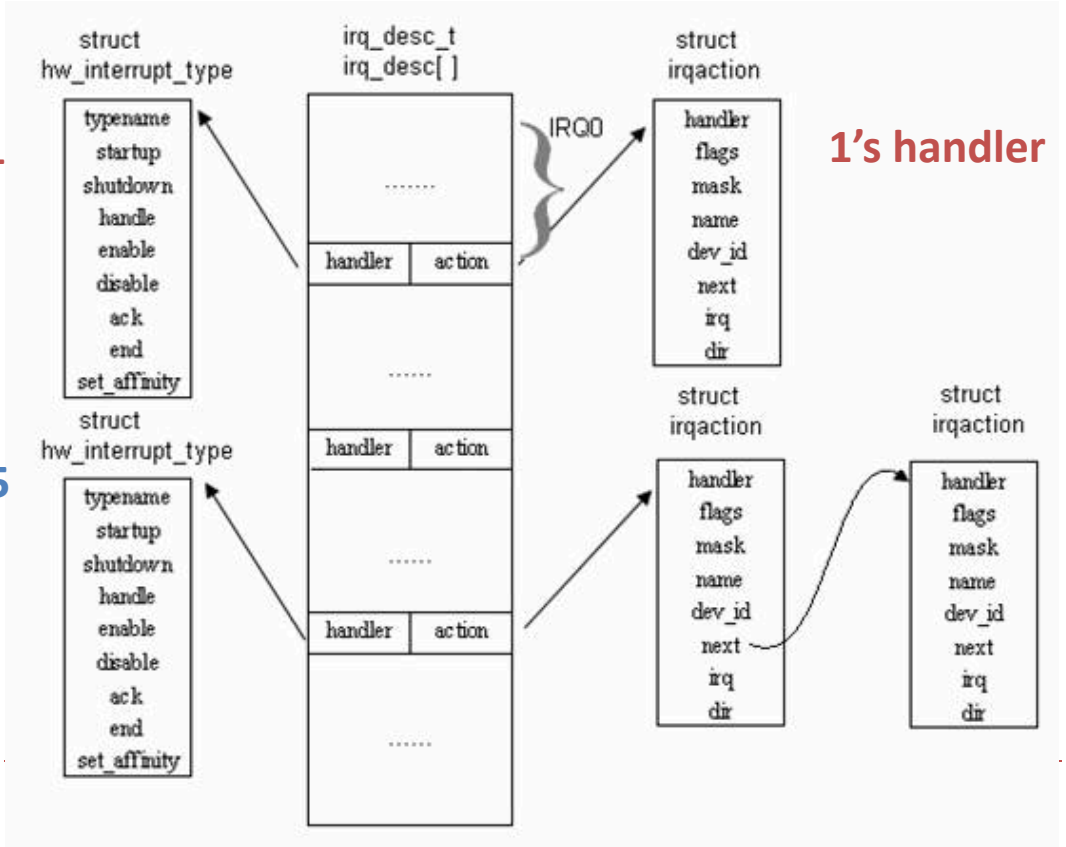
- There are approximately 300 system-calls in Linux 2.6.
- An array of function-pointers (identified by the ID number)
- This array is named 'sys_call_table[]' in Linux https://elixir.bootlin.com/linux/v5.0/source/arch/x86/entry/syscall_64.c



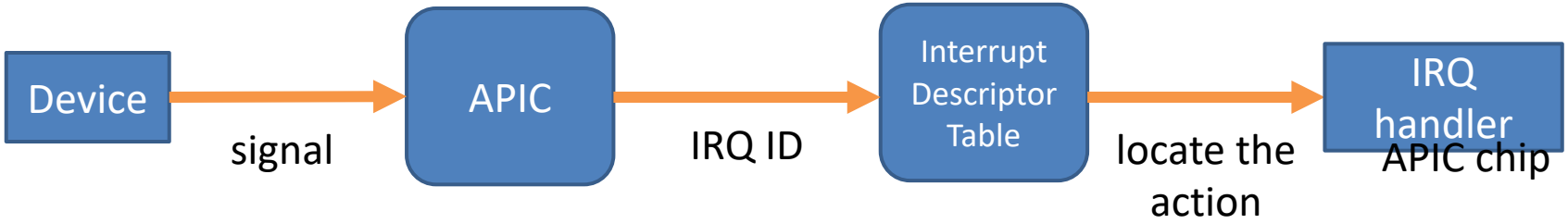
The 'jump-table' idea

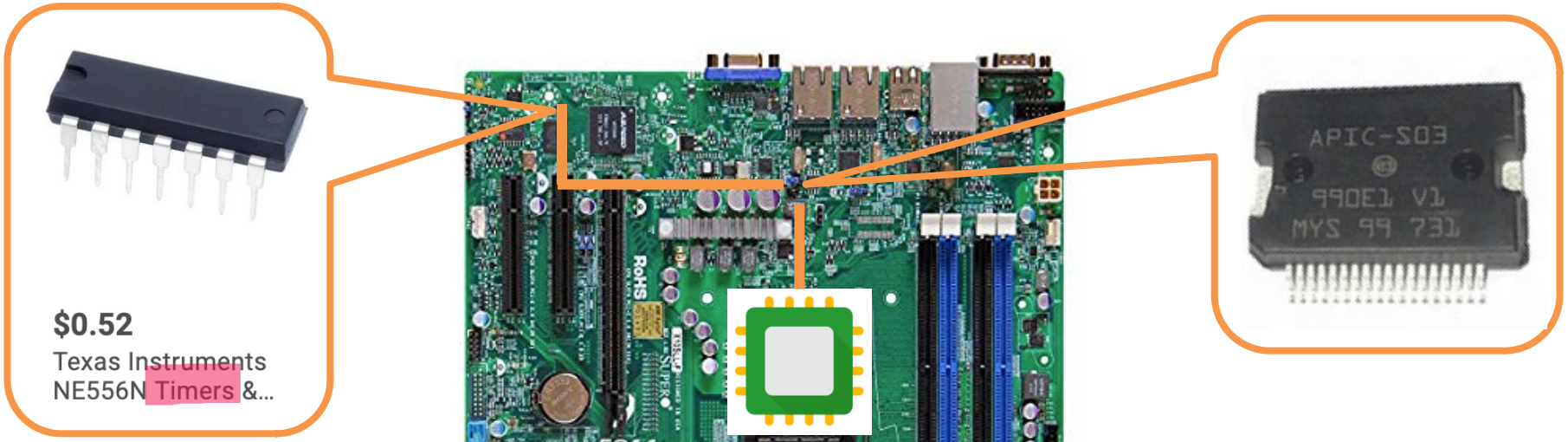
Interrupt Descriptor Table (IDT)

- IDT is in memory, initialized by OS at boot
- IDT associates with each interrupt vector and stores the entry address of the corresponding interrupt handler



The process of interrupt





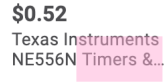
The process of interrupt

Take the ***time interrupt*** as an example

signal

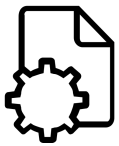
IRQ: 48

```
[ec2-user@ip-172-31-36-115 ~]$ cat /proc/interrupts
           CPU0
  0:         47   IO-APIC   2-edge      timer
  1:          9  xen-pirq   1-ioapic-edge  i8042
  4:       2599  xen-pirq   4-ioapic-edge  ttyS0
  8:          2  xen-pirq   8-ioapic-edge  rtc0
  9:          0  xen-pirq   9-ioapic-level  acpi
 12:         86  xen-pirq  12-ioapic-edge  i8042
 14:          0   IO-APIC  14-edge      ata_piix
 15:          0   IO-APIC  15-edge      ata_piix
 48:      43857  xen-percpu     -virq      timer0
 49:          0  xen-percpu     -ipi       resched0
 50:          0  xen-percpu     -ipi       callfunc0
 51:          0  xen-percpu     -virq      debug0
 52:          0  xen-percpu     -ipi       callfuncsingle0
 53:          0  xen-percpu     -ipi       spinlock0
 54:        254   xen-dyn    -event     xenbus
```

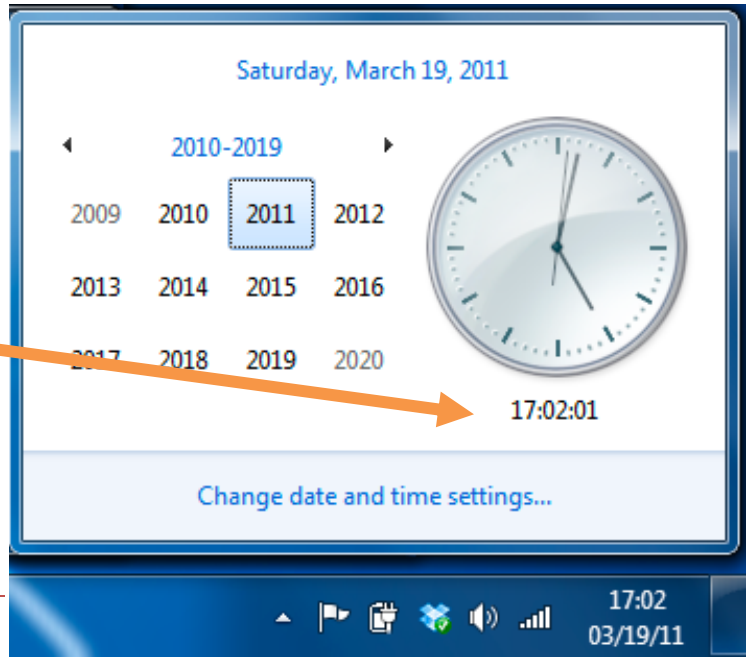


Find timer interrupt in IDT

Timer_handler



Timer ticks



Interrupt vs. Exception vs. System call

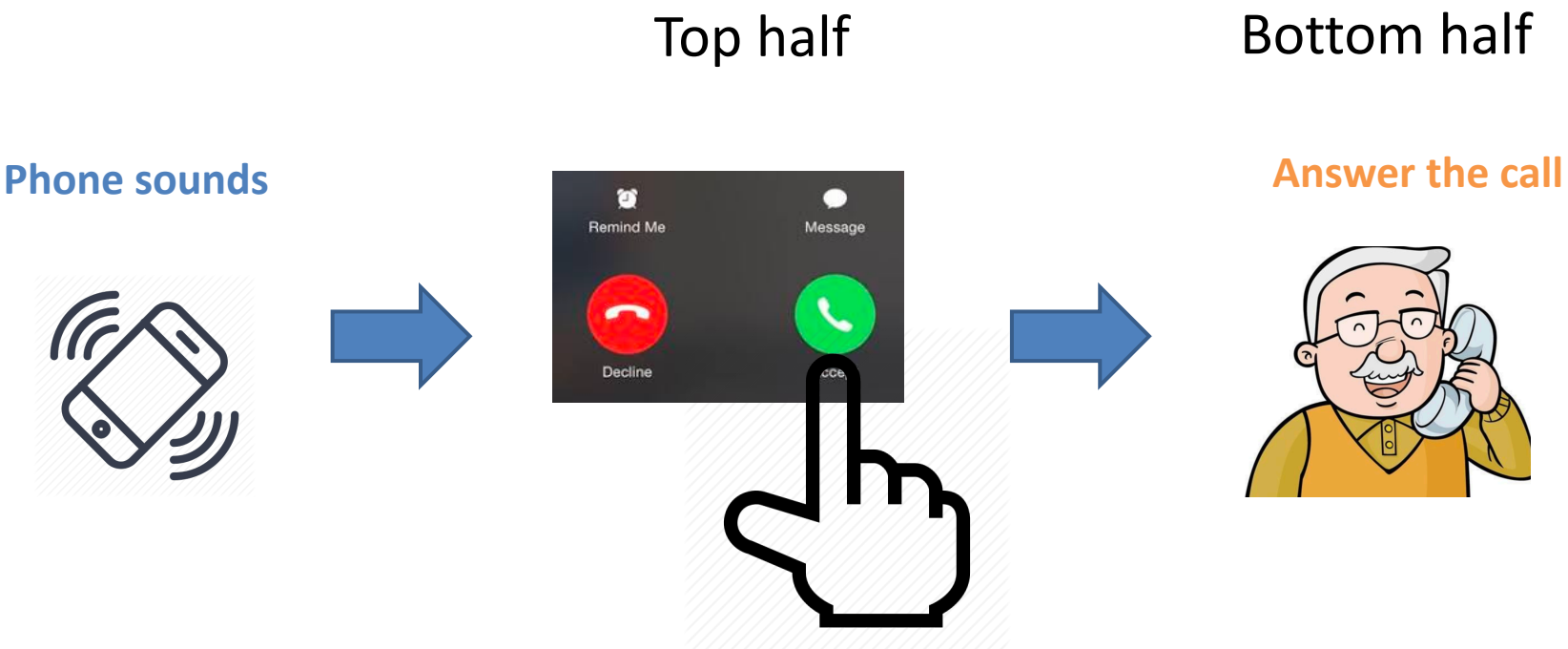
	Source	Handling	Mechanism
Interrupt	Device, etc.	Asynchronous	Interrupts are handled by the processor after finishing the current instruction. If it finds a signal on its interrupt pin, it will look up the address of the interrupt handler in the interrupt table and pass that routine control. After returning from the interrupt handler routine, it will resume program execution at the instruction after the interrupted instruction.
Exception	Application or kernel unexpected behaviors	Synchronous	Exceptions on the other hand are divided into three kinds. These are Faults, Traps and Aborts. Faults are detected and serviced by the processor before the faulting instructions. Traps are serviced after the instruction causing the trap. Aborts are used only to signal severe system problems, when operation is no longer possible.
System call	Application requests	Asynchronous or Synchronous	A way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via API.

Outline

- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity

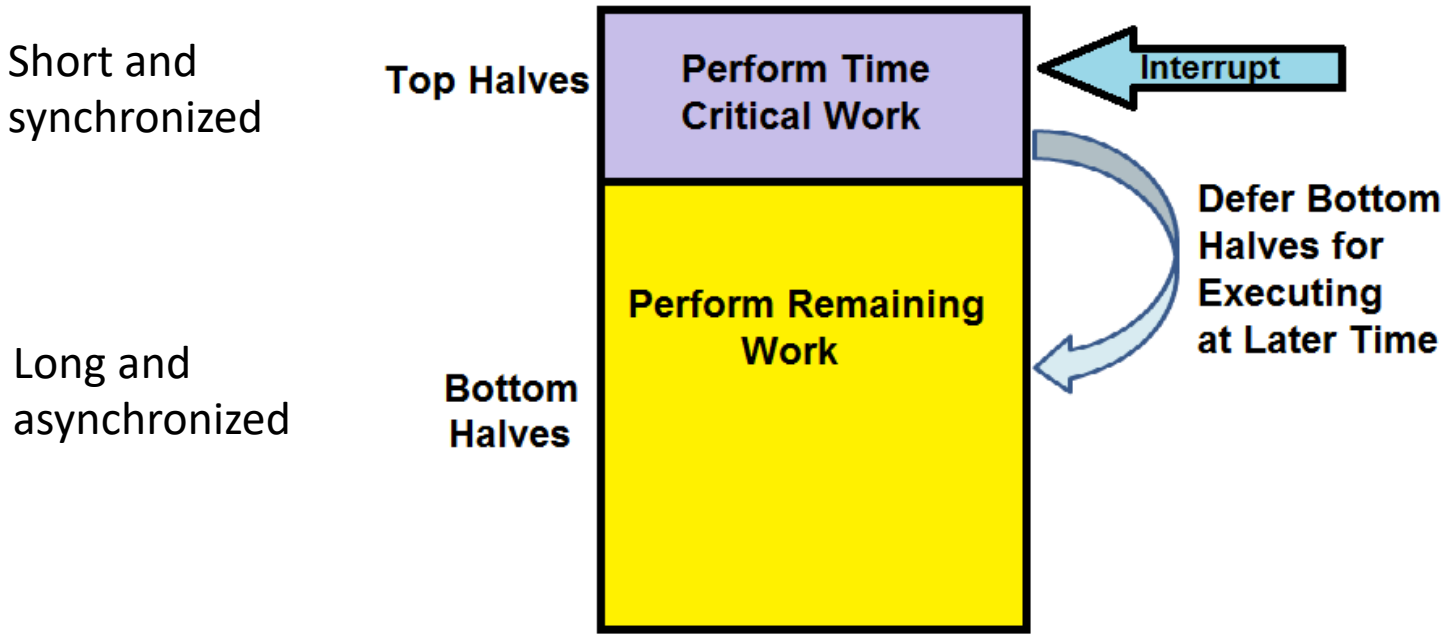
Top-half and bottom-half in interrupts

- Top half: the routine that actually responds to the interrupt
- Bottom half: the routine that processes the interrupt



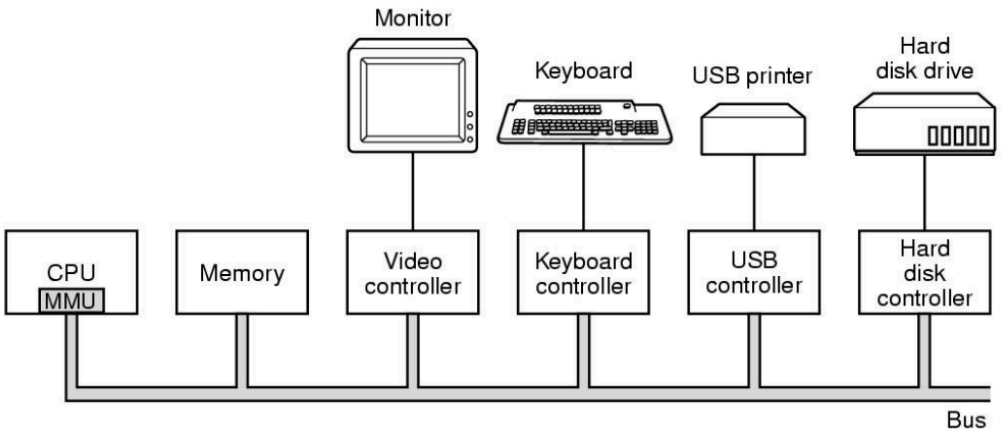
Top-half and bottom-half in interrupts

- Top half: the routine that actually responds to the interrupt
- Bottom half: the routine that processes the interrupt

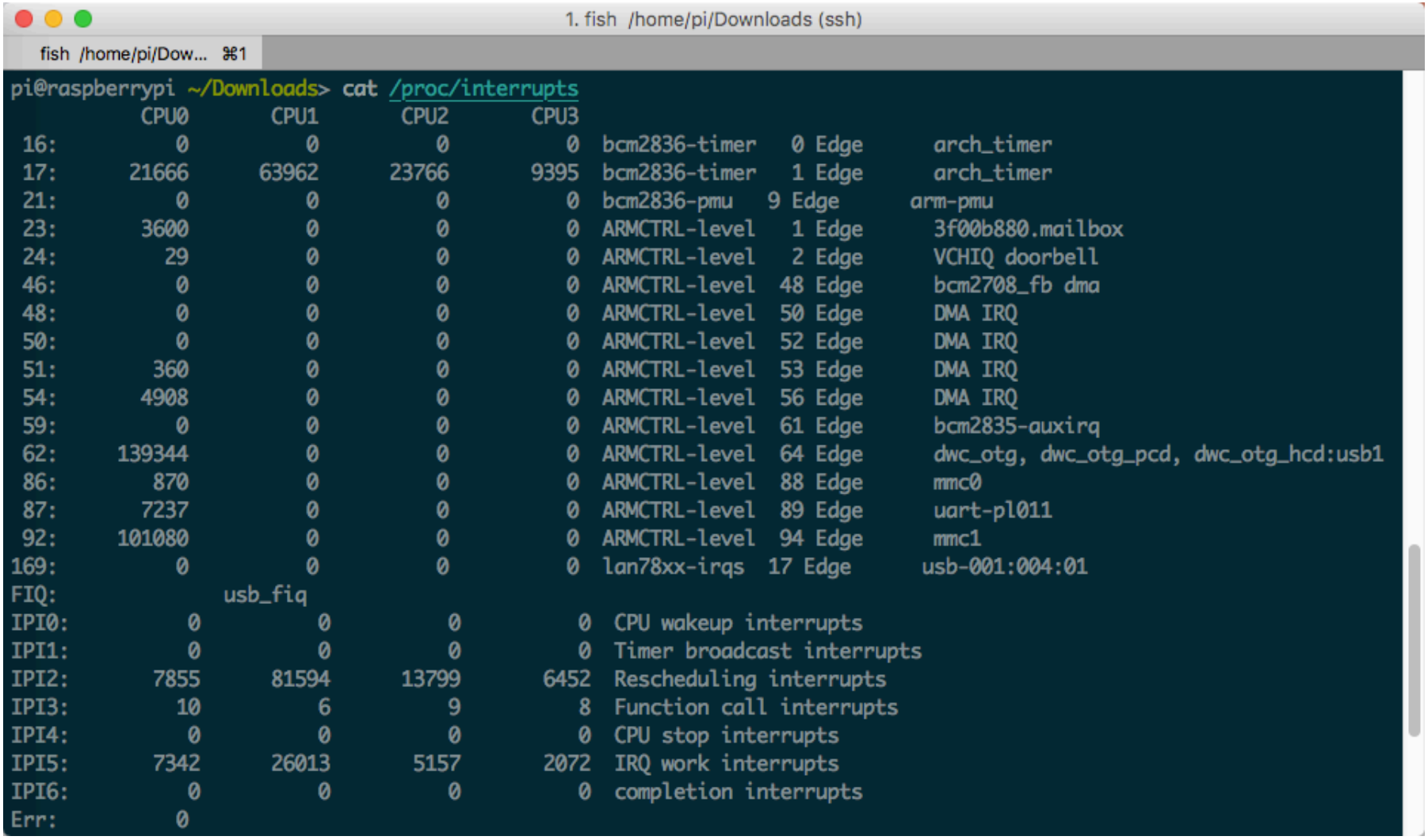


Hardware interrupt

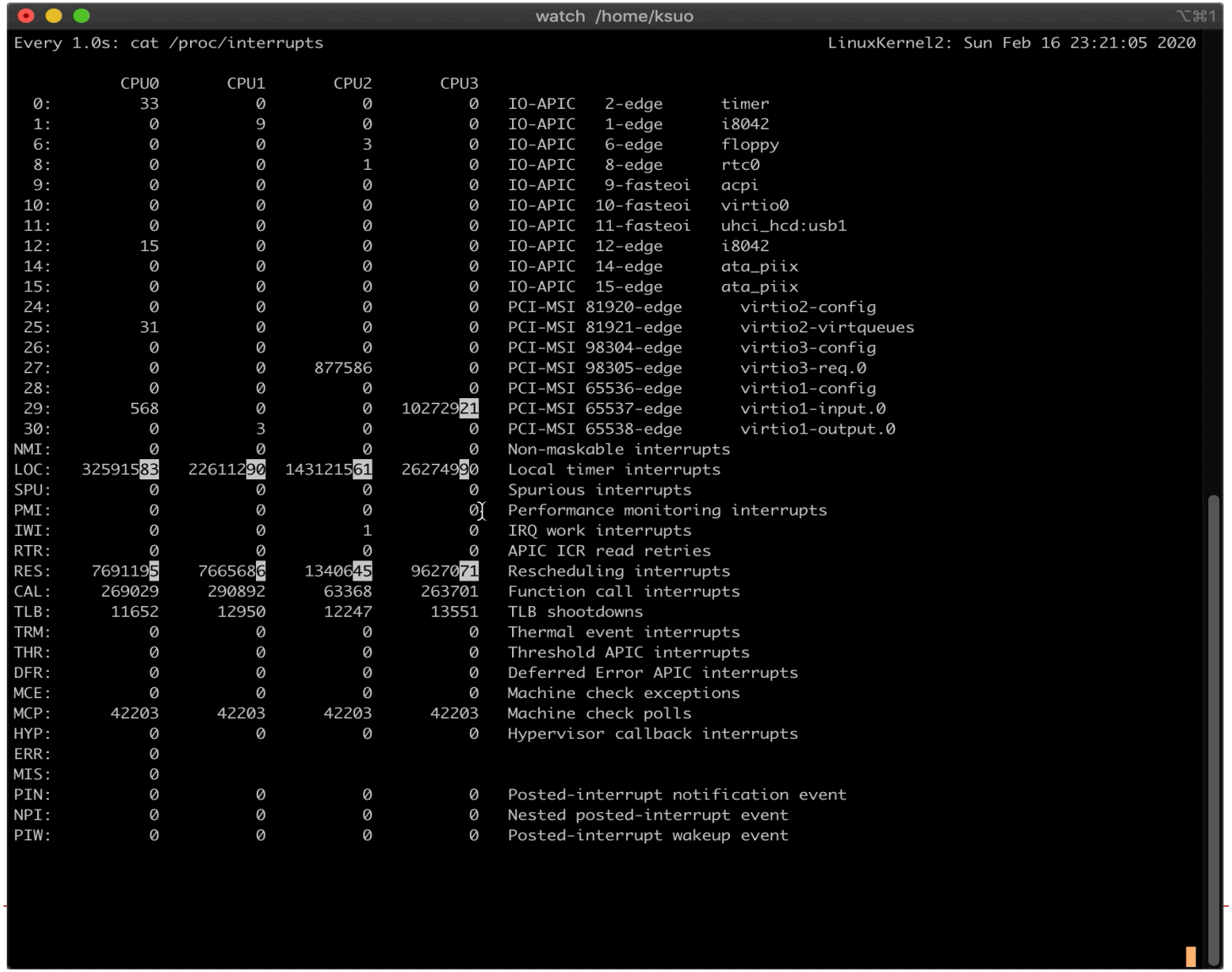
- Hardware interrupts are used to signal that a particular device needs attention:
  - a disk read completed, or
  - a network packet was received, or
  - a timer is ready



Hardware interrupt: cat /proc/interrupts



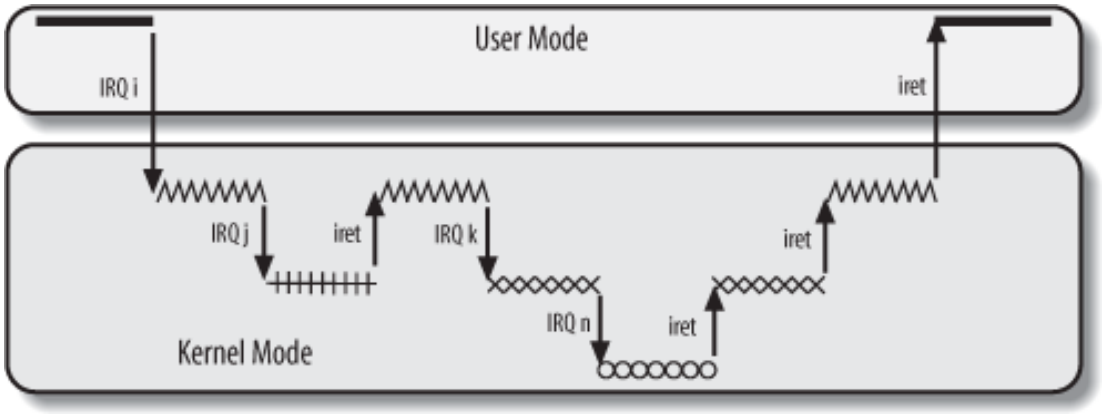
$ watch -n 1 -d cat /proc/interrupts




```
           CPU0       CPU1       CPU2       CPU3
  0:          33          0          0          0   IO-APIC   2-edge      timer
  1:           0          9          0          0   IO-APIC   1-edge      i8042
  6:           0          0          3          0   IO-APIC   6-edge      floppy
  8:           0          0          1          0   IO-APIC   8-edge      rtc0
  9:           0          0          0          0   IO-APIC   9-fasteoi   acpi
 10:           0          0          0          0   IO-APIC  10-fasteoi   virtio0
 11:           0          0          0          0   IO-APIC  11-fasteoi   uhci_hcd:usb1
 12:          15          0          0          0   IO-APIC  12-edge      i8042
 14:           0          0          0          0   IO-APIC  14-edge      ata_piix
 15:           0          0          0          0   IO-APIC  15-edge      ata_piix
 24:           0          0          0          0   PCI-MSI 81920-edge      virtio2-config
 25:          31          0          0          0   PCI-MSI 81921-edge      virtio2-virtqueues
 26:           0          0          0          0   PCI-MSI 98304-edge      virtio3-config
 27:           0          0     877586          0   PCI-MSI 98305-edge      virtio3-req.0
 28:           0          0          0          0   PCI-MSI 65536-edge      virtio1-config
 29:         568          0          0   10272921   PCI-MSI 65537-edge      virtio1-input.0
 30:           0          3          0          0   PCI-MSI 65538-edge      virtio1-output.0
NMI:           0          0          0          0   Non-maskable interrupts
LOC:    32591583   22611290   14312156   26274990   Local timer interrupts
SPU:           0          0          0          0   Spurious interrupts
PMI:           0          0          0          0   Performance monitoring interrupts
IWI:           0          0          1          0   IRQ work interrupts
RTR:           0          0          0          0   APIC ICR read retries
RES:     7691195    7665686    1340645    9627071   Rescheduling interrupts
CAL:      269029     290892      63368     263701   Function call interrupts
TLB:       11652      12950      12247      13551   TLB shootdowns
TRM:           0          0          0          0   Thermal event interrupts
THR:           0          0          0          0   Threshold APIC interrupts
DFR:           0          0          0          0   Deferred Error APIC interrupts
MCE:           0          0          0          0   Machine check exceptions
MCP:       42203      42203      42203      42203   Machine check polls
HYP:           0          0          0          0   Hypervisor callback interrupts
ERR:           0
MIS:           0
PIN:           0          0          0          0   Posted-interrupt notification event
NPI:           0          0          0          0   Nested posted-interrupt event
PIW:           0          0          0          0   Posted-interrupt wakeup event
```

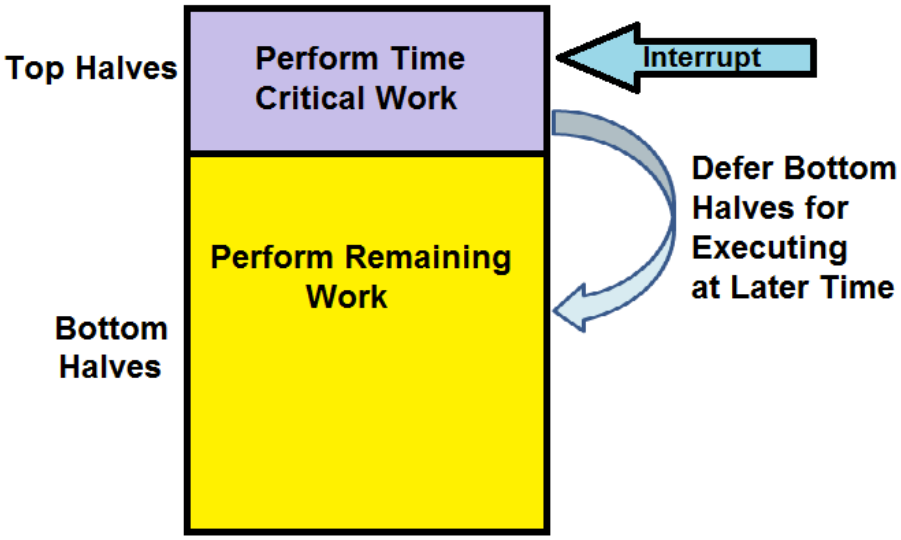
Hardware interrupt feature

- **Definition:** raised by hardware
- **Nest:** hardirq in Linux can be nested
- **Interrupt controller:** IRQ number is provided by APIC



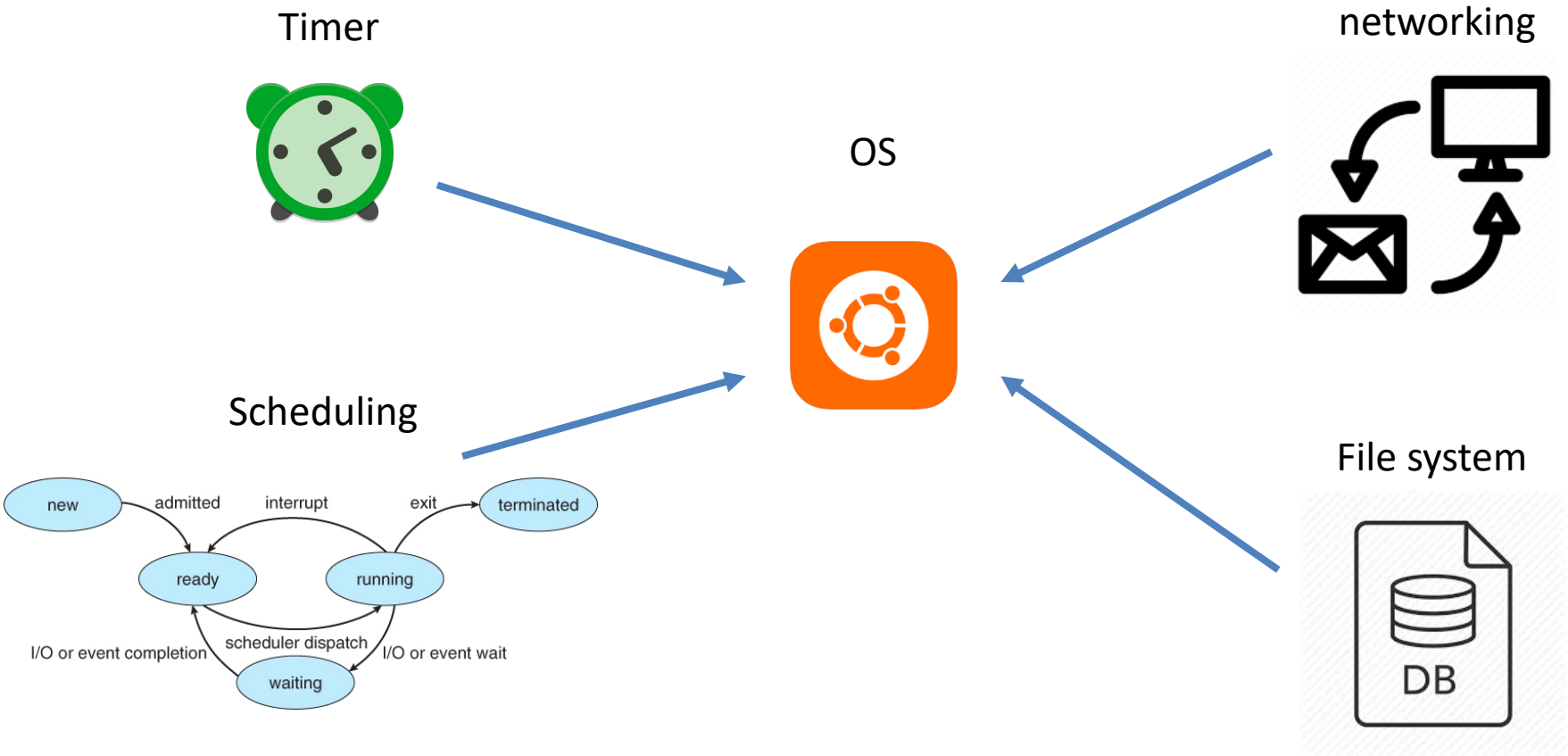
Hardware interrupt feature

- **Mask:** hardirq can be masked
- **Top/Bottom:** hardirq handler (small) ensures that it completes the task quickly
- **Preemption:** hardirq has priority and can preempt over softirq



Software interrupt feature

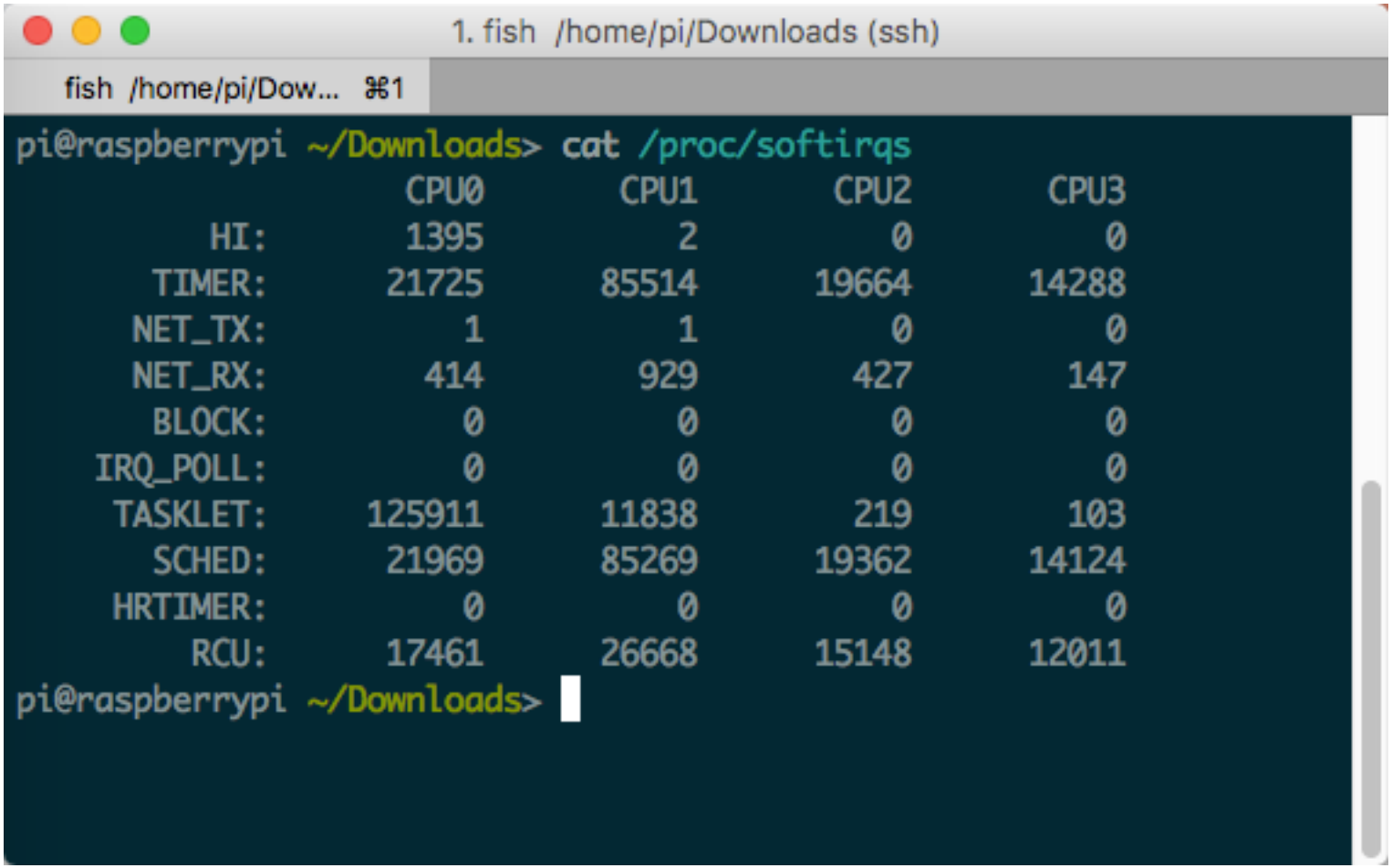
- **Definition:** raised by software or execution



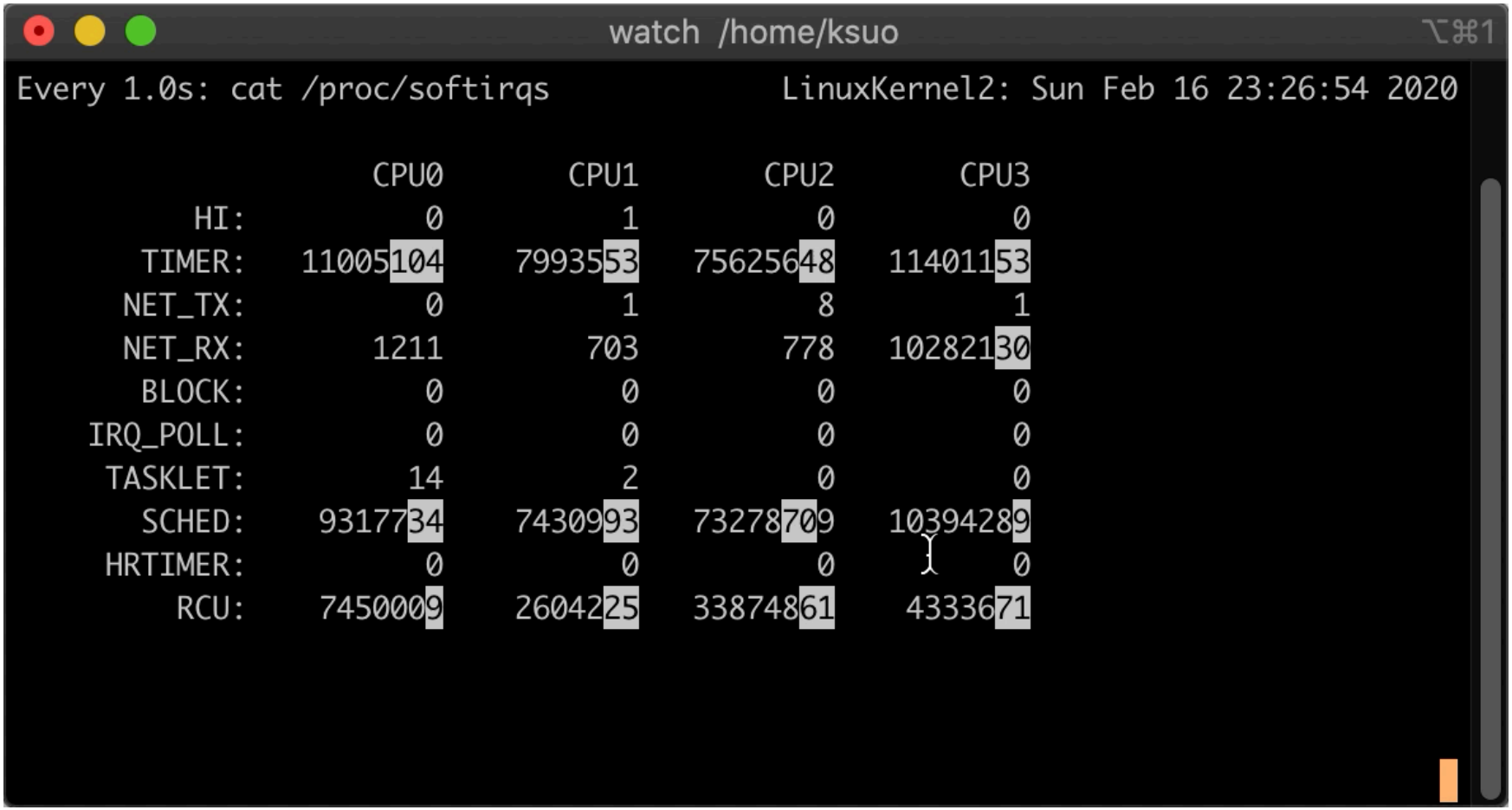
Software interrupt feature

- **Definition:** raised by software or execution
- **Nest:** softirq in Linux cannot be nested
- **Interrupt controller:** softirq does not have IRQ number
- **Mask:** softirq cannot be masked
- **Top/Bottom:** softirq is responsible the bottom-half work of the interrupt
- **Preemption:** softirq cannot preempt with each other

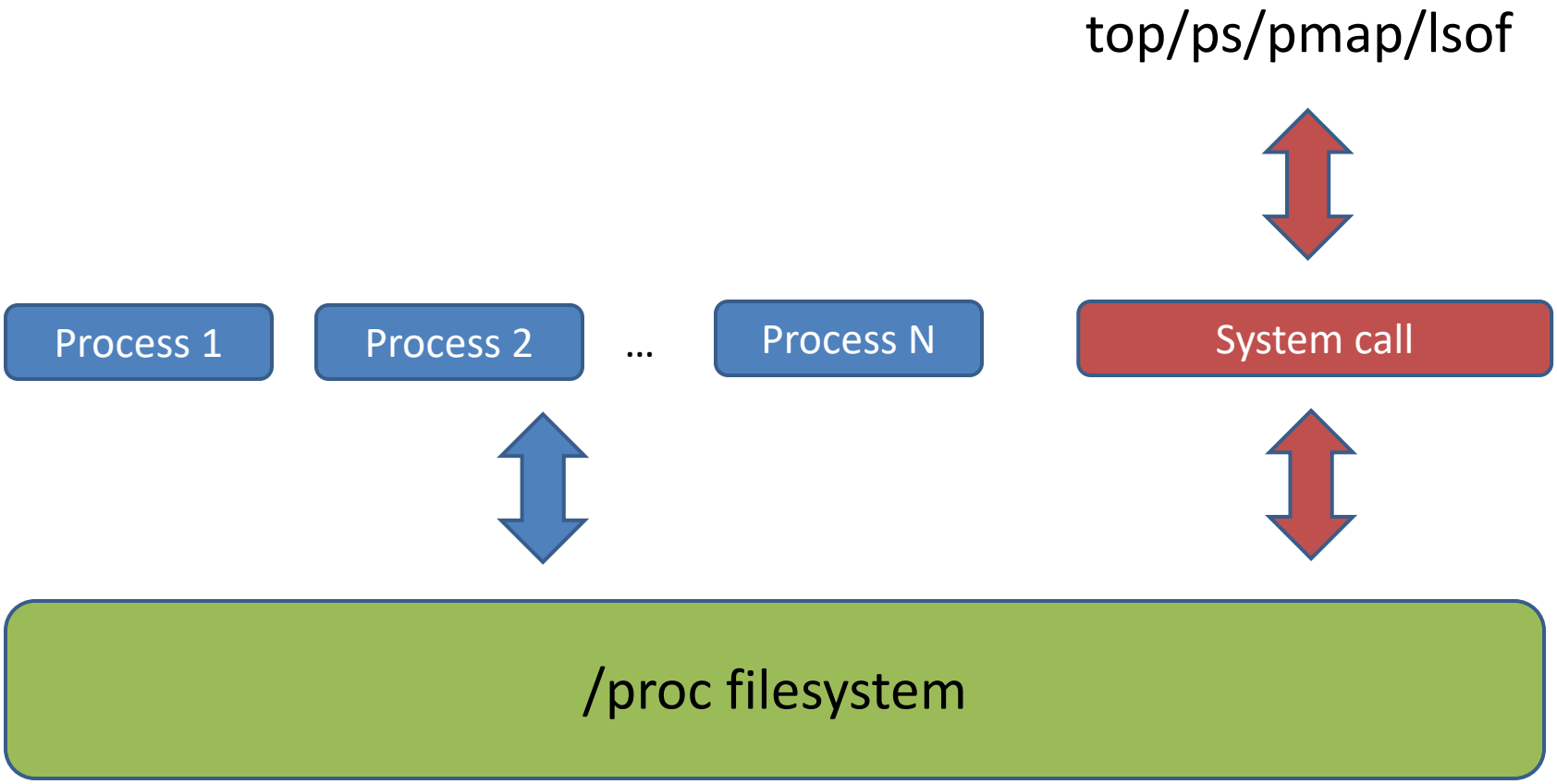
Software interrupt: cat /proc/softirqs



$ watch -n 1 -d cat /proc/softirqs

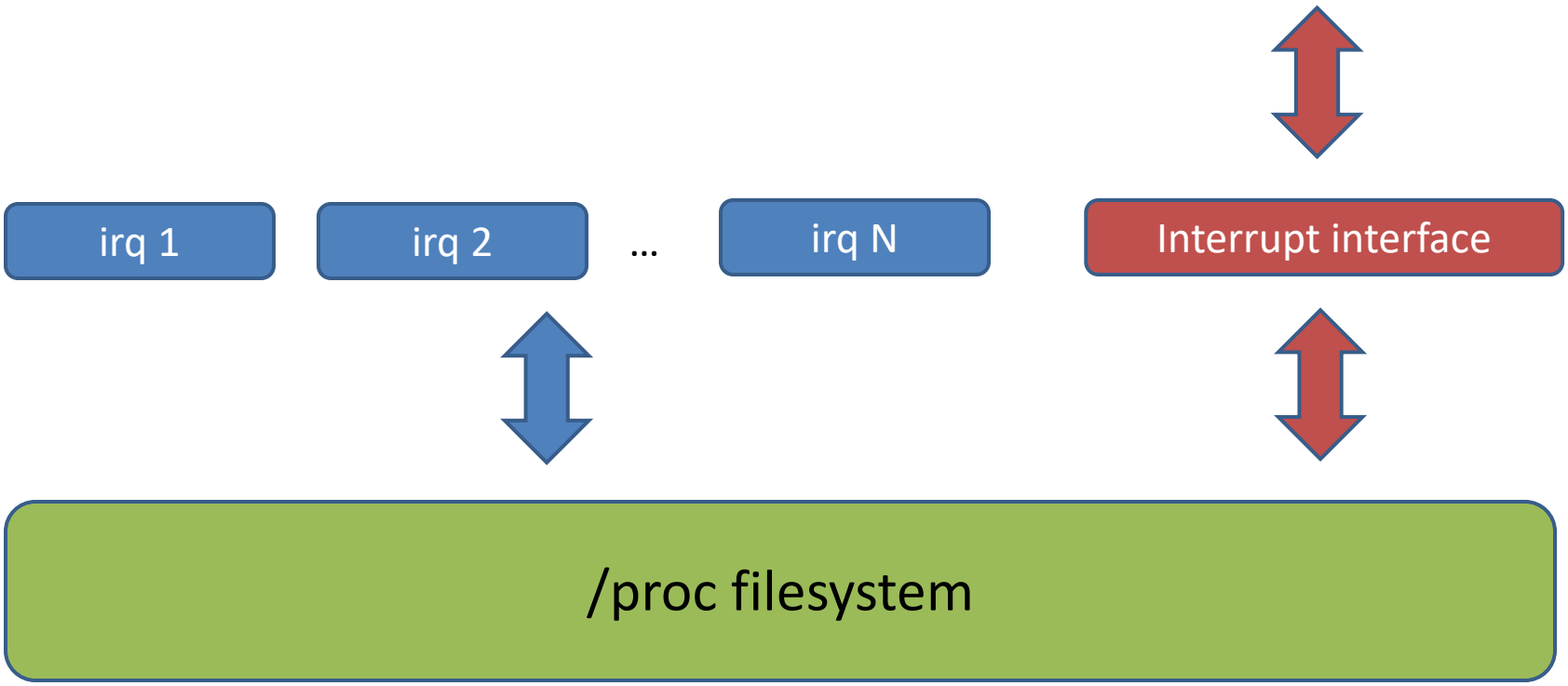


System call: the /proc/ filesystem



Interrupt: the /proc/ filesystem

cat /proc/interrupts
cat /proc/softirqs

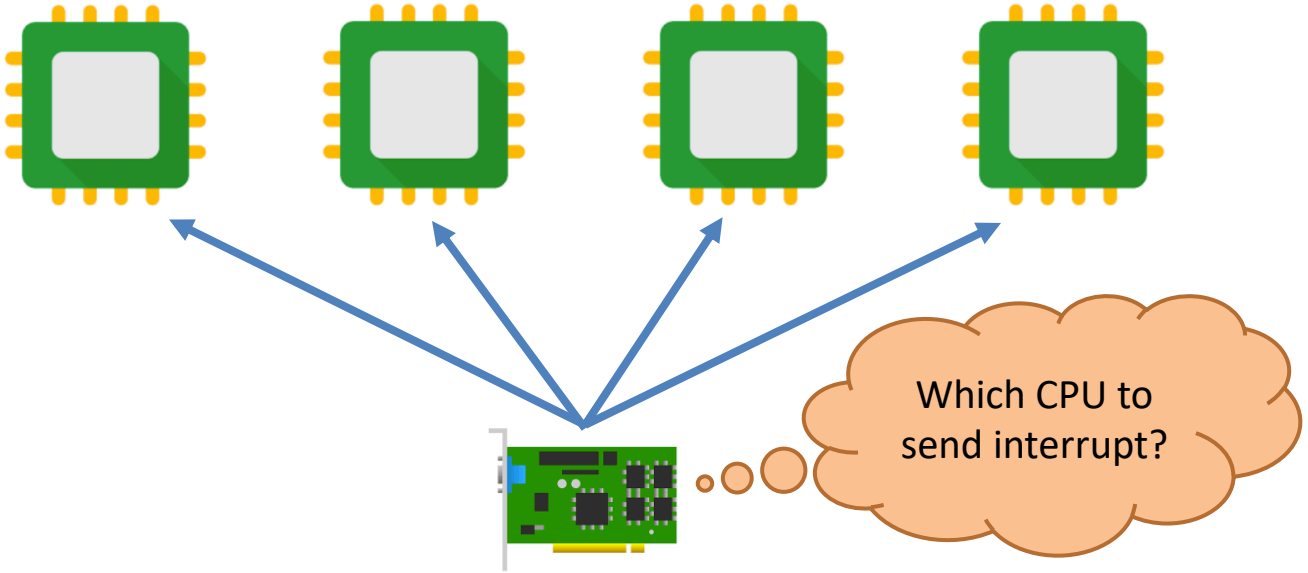


Hardware IRQ vs. Software IRQ

	Hardware IRQ	Software IRQ
Definition	raised by hardware	raised by software or execution
Nest	can be nested	cannot be nested
Interrupt controller	IRQ number is provided by APIC	softirq does not have IRQ number
Mask	can be masked	cannot be masked
Top/Bottom	hardirq handler ensures that it completes the task quickly	softirq is responsible the bottom-half work of the interrupt
Preemption	hardirq has priority and can preempt over softirq	softirq cannot preempt with each other

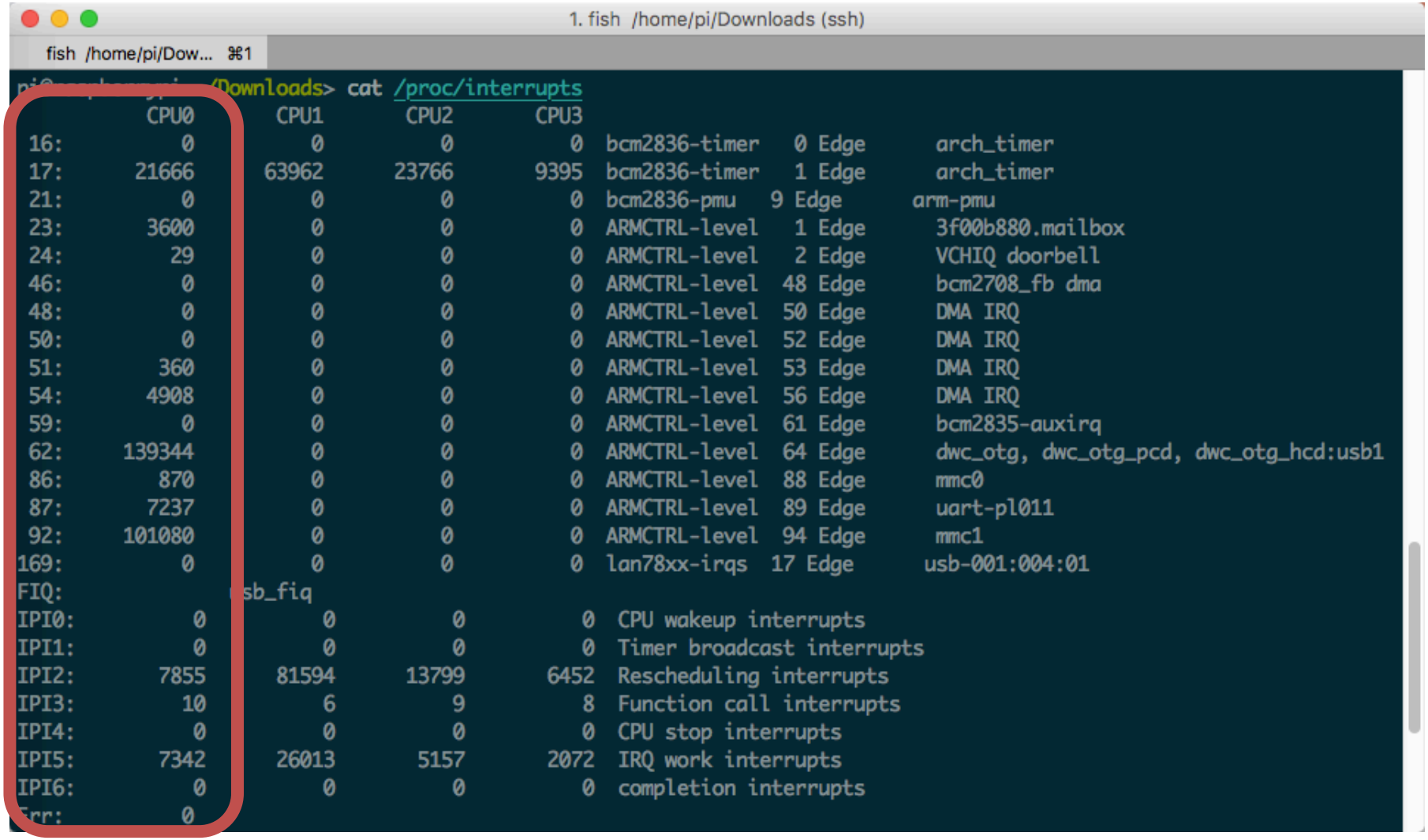
Interrupt Affinity

- IRQ affinity determines the CPU cores that are allowed to execute the processing for that IRQ
- IRQ affinity can be used to improve application performance (e.g., for multi CPU system like NUMA)



Interrupt Affinity

watch -n 1 -d cat /proc/interrupts



1. nmon /home/ksuo (ssh)

iperf3 / (ssh) ⌘1 | iperf3 / (ssh) ⌘2 | fish /home/ks... ⌘3 | fish /home/ks... ⌘4 | nmon /home/k... ⌘5 | fish /Users/ton... ⌘6

```
nmon-16f-------------------Hostname=ksuo-2--------Refresh= 2secs ---16:07.40
 CPU Utilisation Stats
ALL    13.0    0.0   17.0  271.4   16.0    0.0   77.0    0.0    0.0    0.0
CPU   User%  Nice%   Sys%  Idle%  Wait% HWirq% SWirq% Steal% Guest% GuestNice%
  1     3.5    0.0    1.0   78.0   16.0    0.0    1.0    0.5    0.0    0.0
  2     5.5    0.0    0.0   93.0    0.0    0.0    0.0    0.0    0.0    0.0
  3     0.5    0.0   15.0    5.5    0.0    0.0   76.0    0.0    0.0    0.0
  4     4.0    0.0    0.5   95.0    0.0    0.0    0.0    0.0    0.0    0.0
```

2. cpubars /home/ksuo (ssh)

cpubars /home/ks... ⌘1

nice user sys iowait irq softirq load: 1.69 1.31 0.82

CPU utilization

CPU cores

```
$ wget http://sourceforge.net/projects/nmon/files/nmon16j.tar.gz
$ tar zxvf nmon16j.tar.gz
$ sudo cp nmon_AMD64_ubuntu1804 /usr/bin/nmon
$ nmon
Press 'U'
```

```
$ sudo apt-get install libncurses5-dev git libncursesw5-dev
$ git clone https://github.com/kevinsuo/cpubars.git
$ cd cpubars
$ make
$ ./cpubars
```

Interrupt Affinity

- Read the IRQ affinity of specific hardIRQ

f = 1111, means all 4 CPU cores will receive the interrupt

```
# cat /proc/irq/32/smp_affinity
f
```

- Set the IRQ affinity to specific hardIRQ

Interrupt from IRQ32 will only go to CPU core 1

```
# echo 1 >/proc/irq/32/smp_affinity
# cat /proc/irq/32/smp_affinity
1
```

- SoftIRQ will execute on the core which it is raised and cannot be set with affinity
  - SoftIRQ can be scaled with RPS/RFS in the kernel settings (irq migration)

Conclusion

- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity