

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

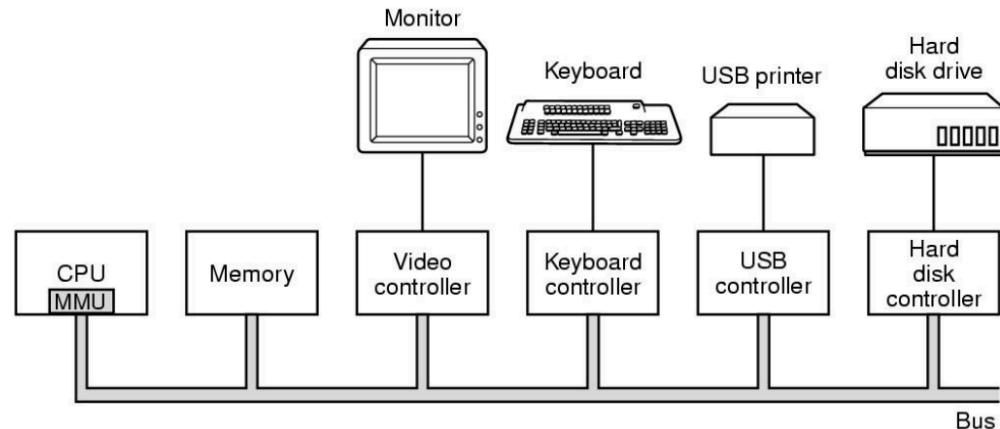


How can the processor work with hardware



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

...



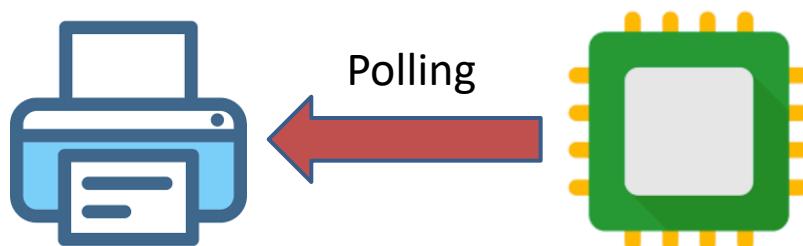
What is Polling?

- 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:

Randomly Every Ns/min

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

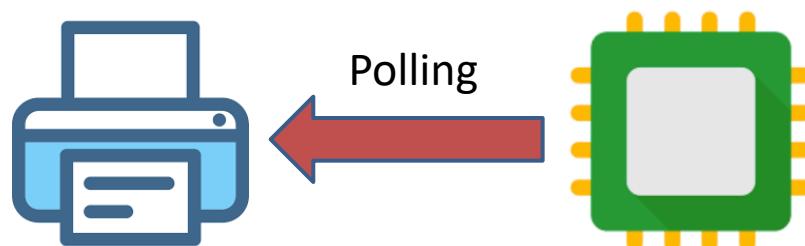


What is Polling?

- 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:

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.

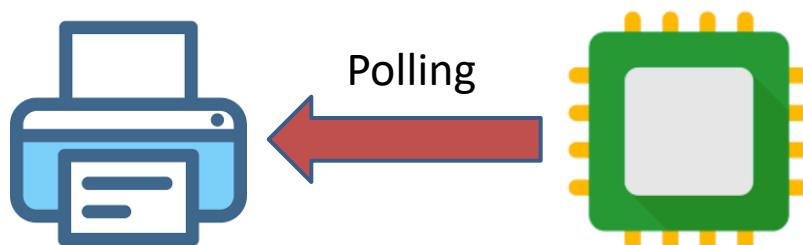


What is Polling?

- 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:

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.

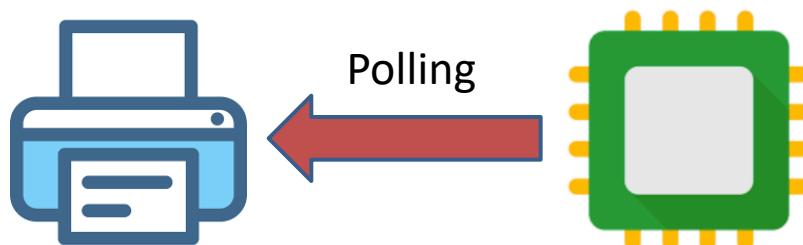


What is Polling?

- 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



Polling

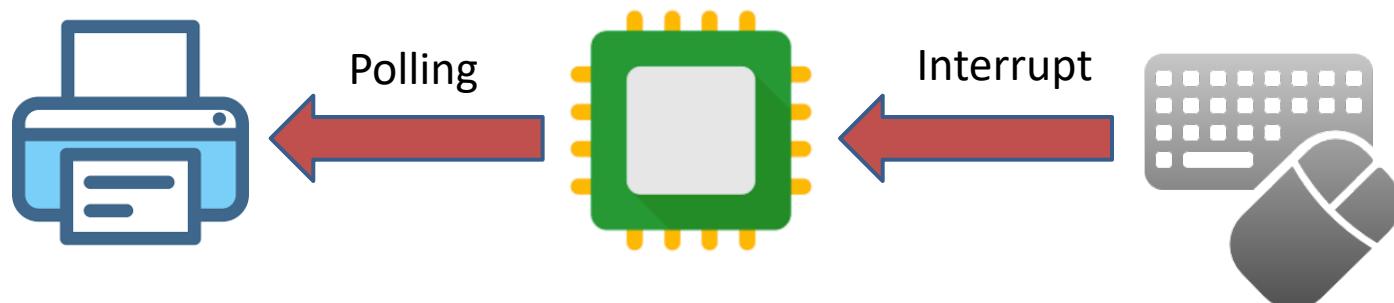


What is interrupt?

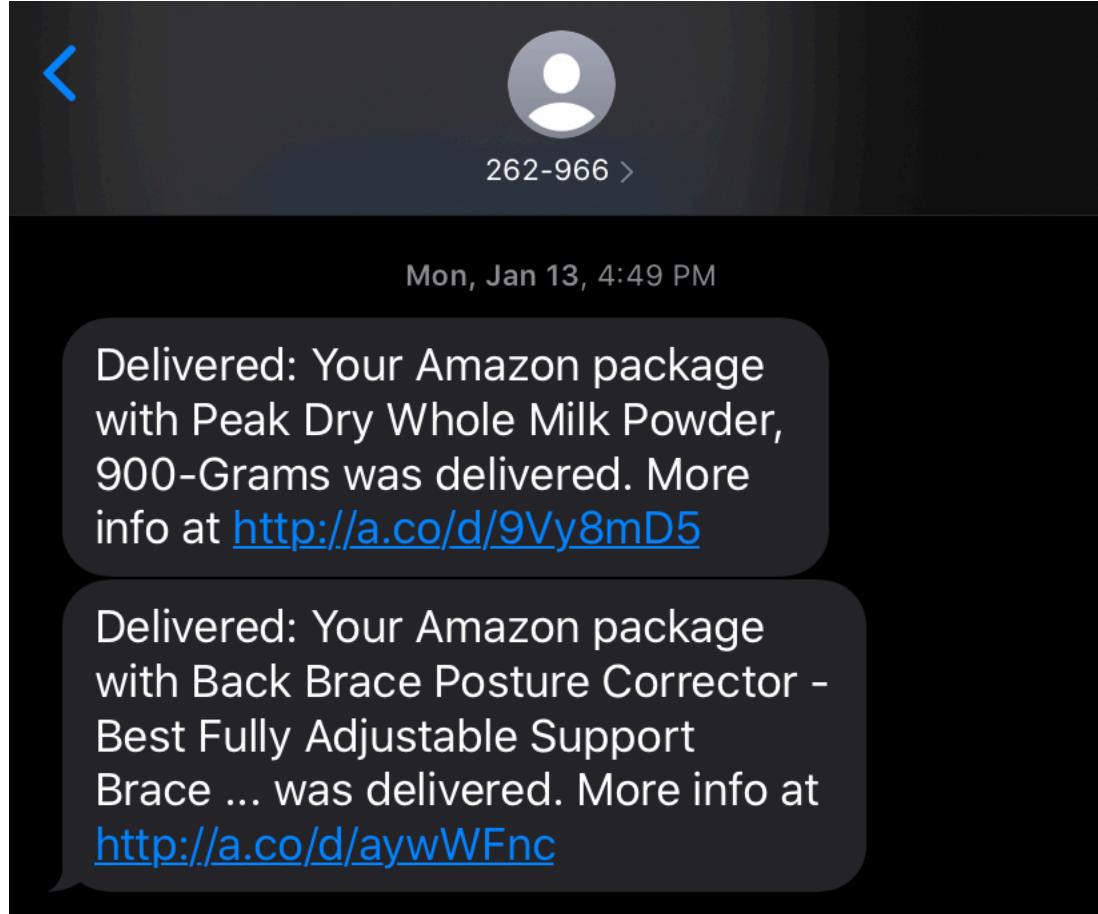
- How can the processor work with hardware without impacting the machine's overall performance?
 - Polling: repeatedly check whether the hardware is ready or not
 - Interrupt: hardware signals to the kernel when attention is needed

Polling:

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



Interrupt



Interrupt vs. Polling

Example for Polling and Interrupt



CPU (Mario)



I/O Device (Princess Peach)

Unregistered PowerVideoMaker

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

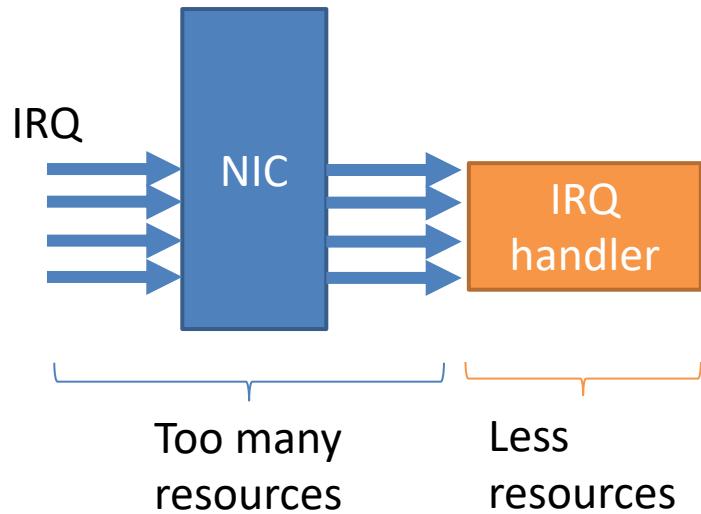


What will happen if you have too many phone calls?



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

Interrupt problem



Interrupt:

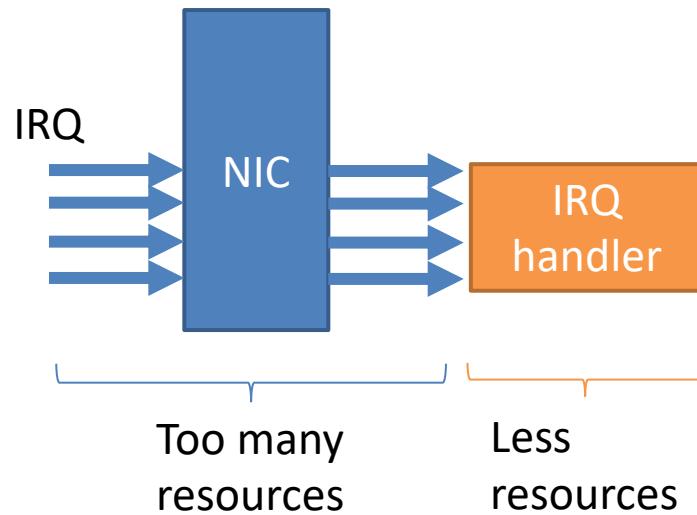
- Too many **interrupt**
- > Less resources for **IRQ handler**
- > miss some interrupt and data

Phone sounds

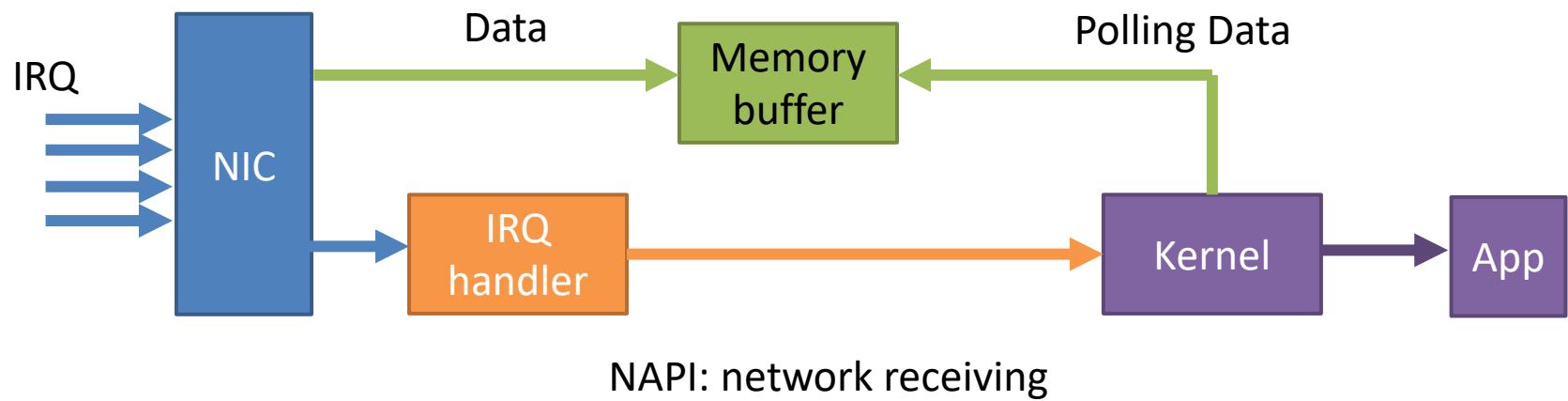
Answer the call



Combine Interrupt & Polling



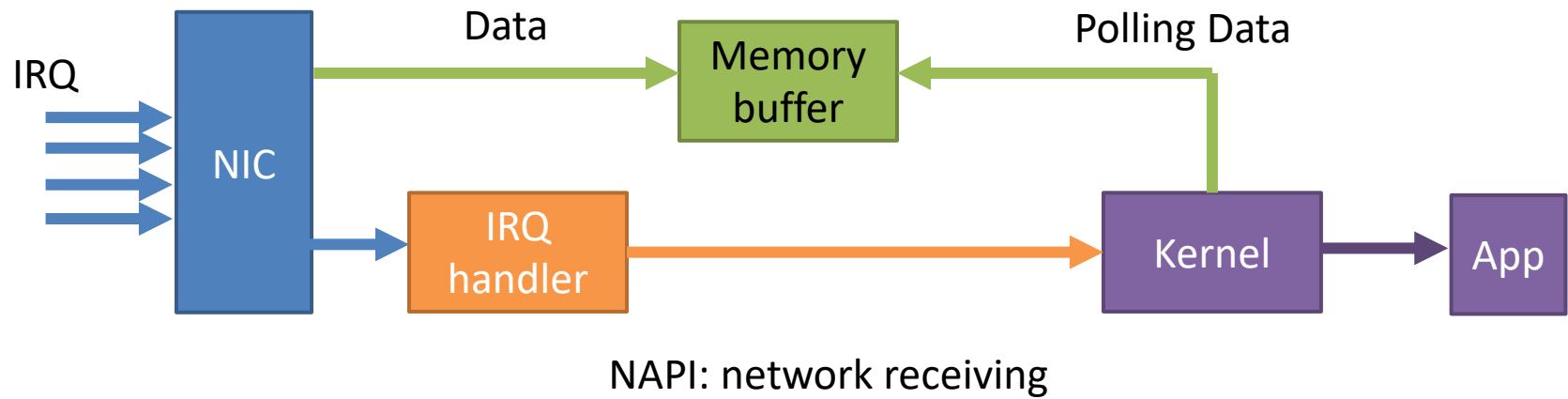
Interrupt:
Too many **interrupt**
-> Less resources for **IRQ handler**
-> miss some interrupt and data



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

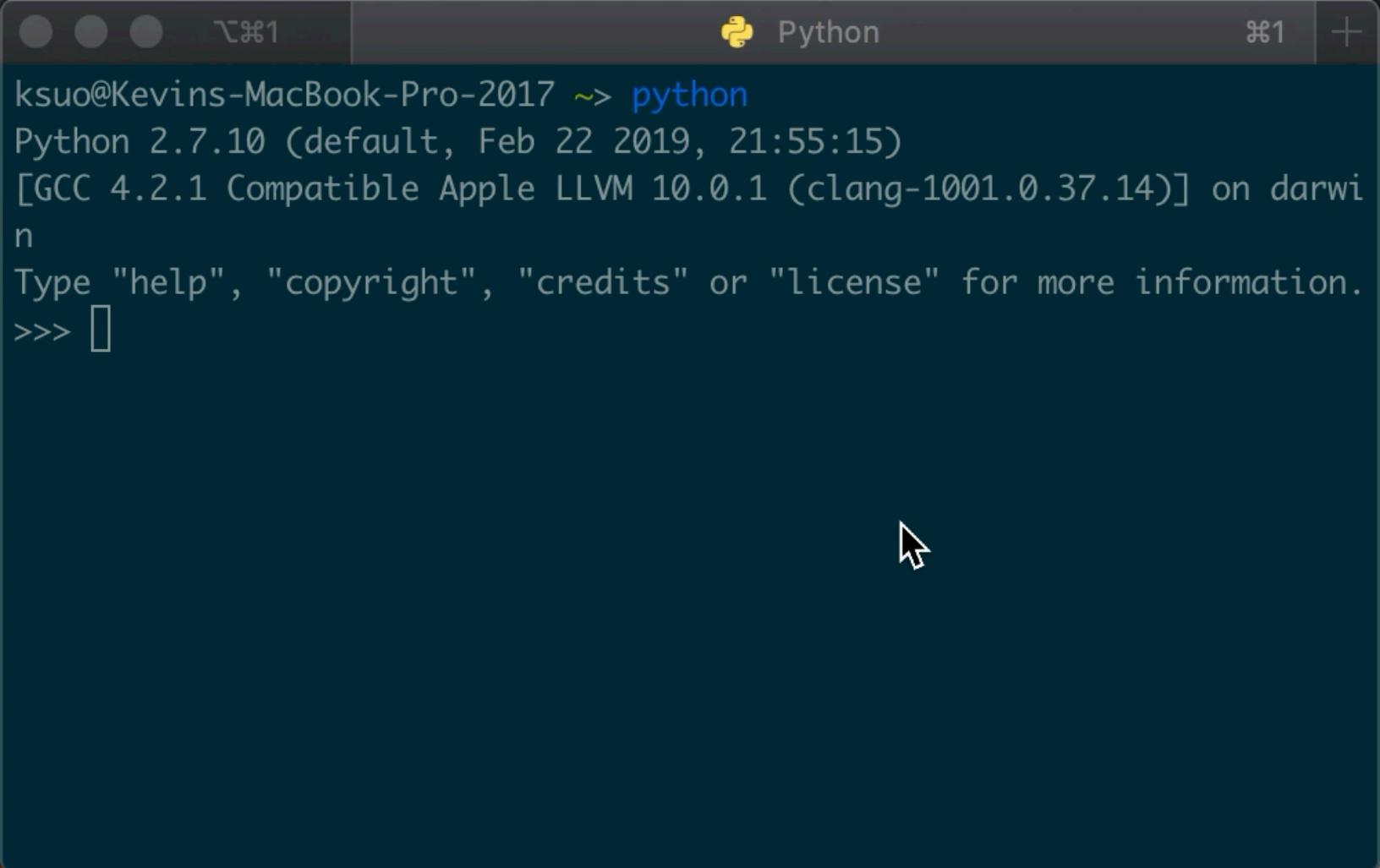


Exceptions

- 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



A screenshot of a macOS terminal window titled "Python". The window shows a Python 2.7.10 session. The output is as follows:

```
ksuo@Kevins-MacBook-Pro-2017 ~> python
Python 2.7.10 (default, Feb 22 2019, 21:55:15)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwi
n
Type "help", "copyright", "credits" or "license" for more information.
>>> []
```

The cursor is visible at the bottom center of the terminal window.



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."
"

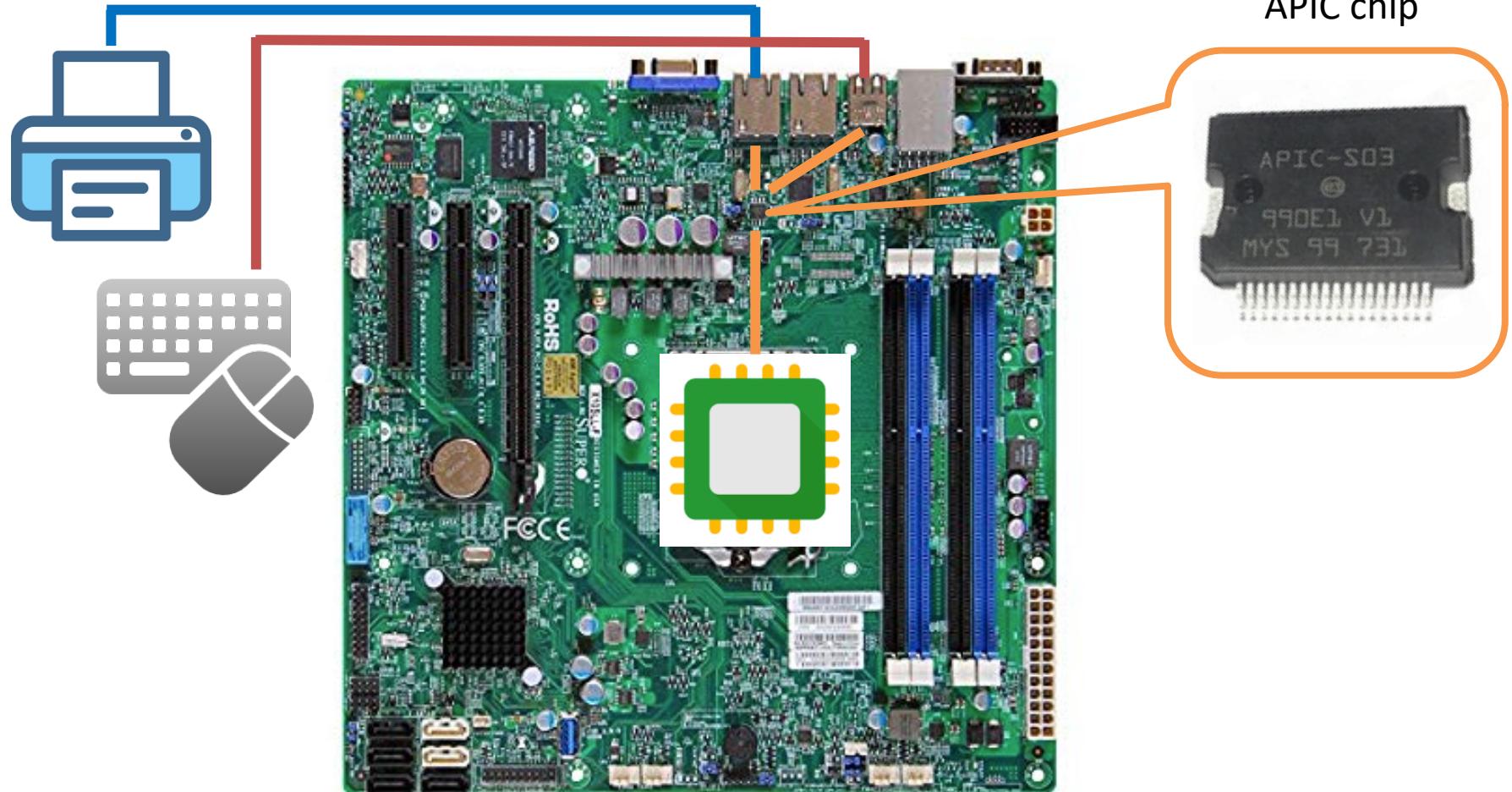


Outline

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

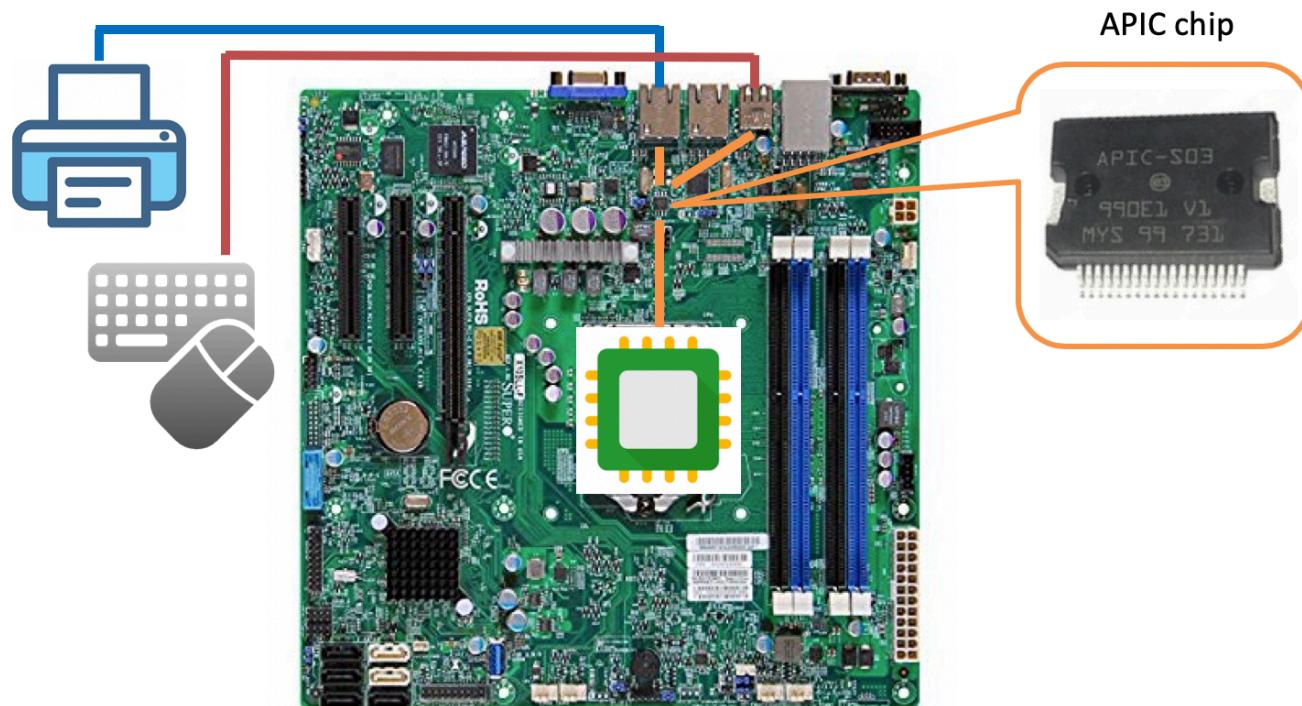


APIC: Advanced programmable interrupt controller



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

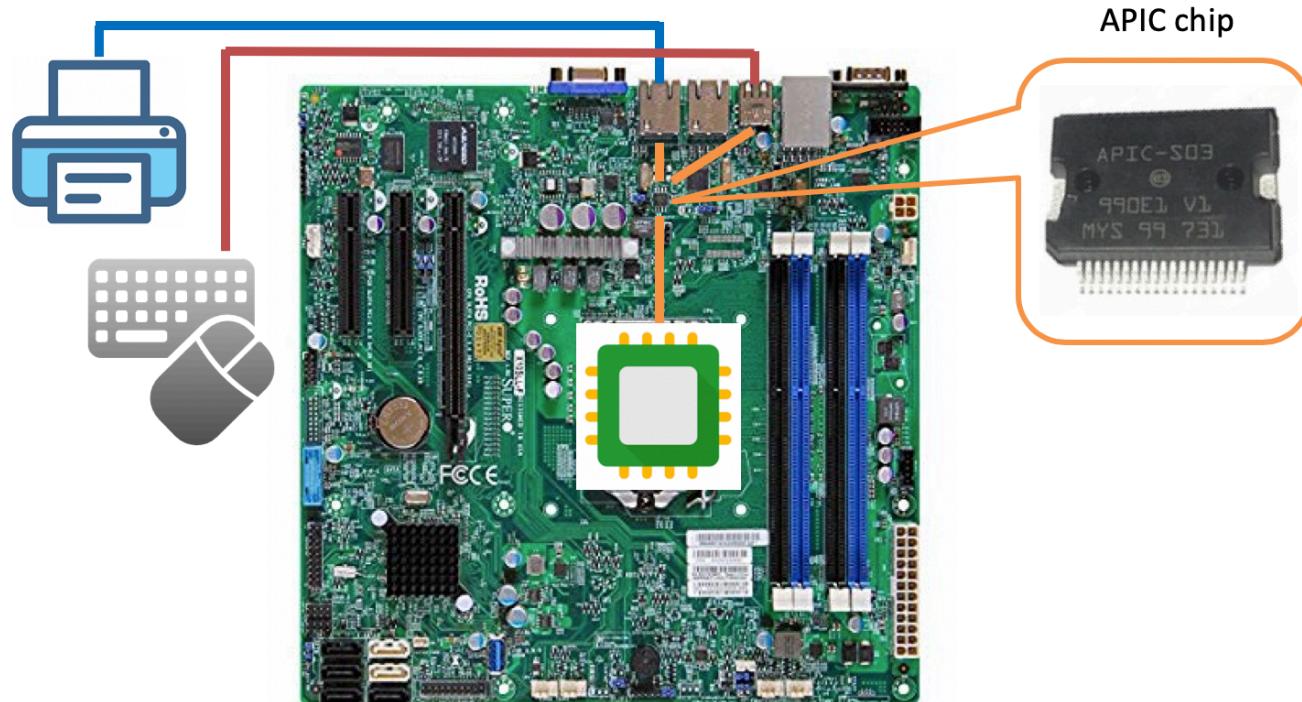


APIC: Advanced program

2. APIC translates IRQ to interrupt number

- Raises interrupt to CPU
- Interrupt # available in register

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 RTL8120/810v Family Fast Ethernet NIC	OK

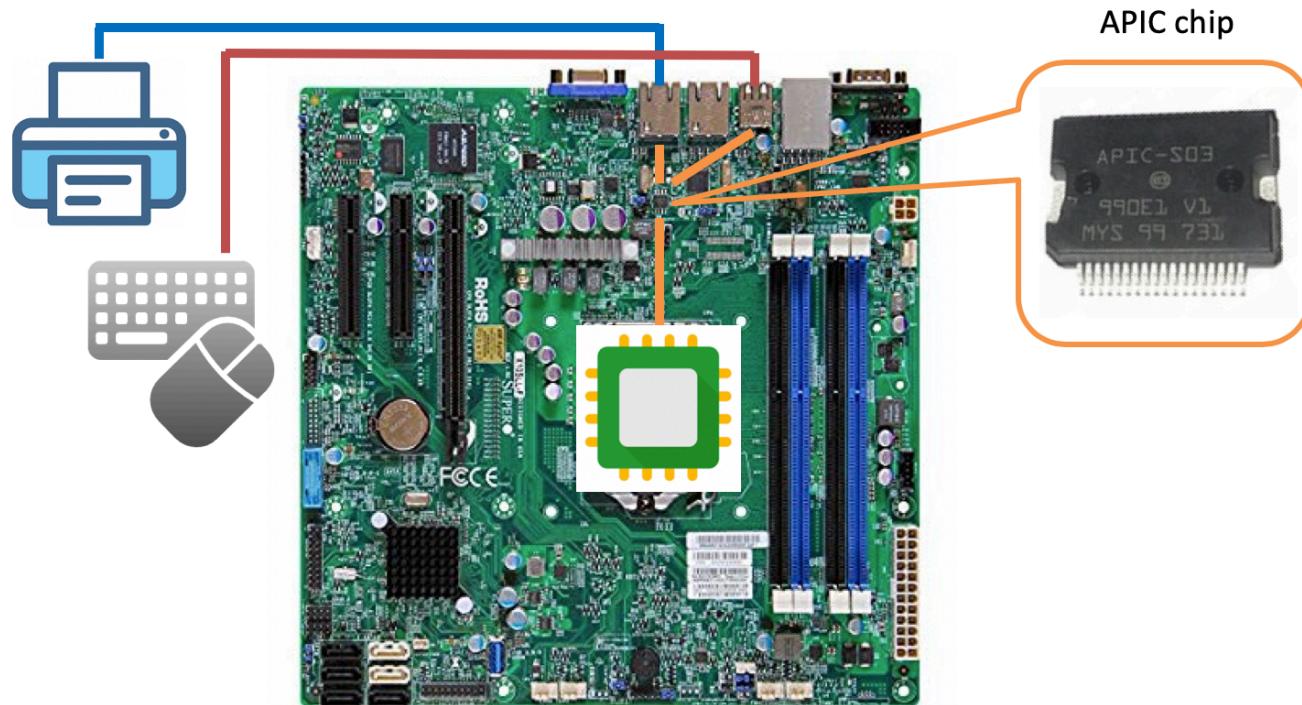


APIC: Advanced programmable interrupt controller

3. Interrupts can have varying priorities

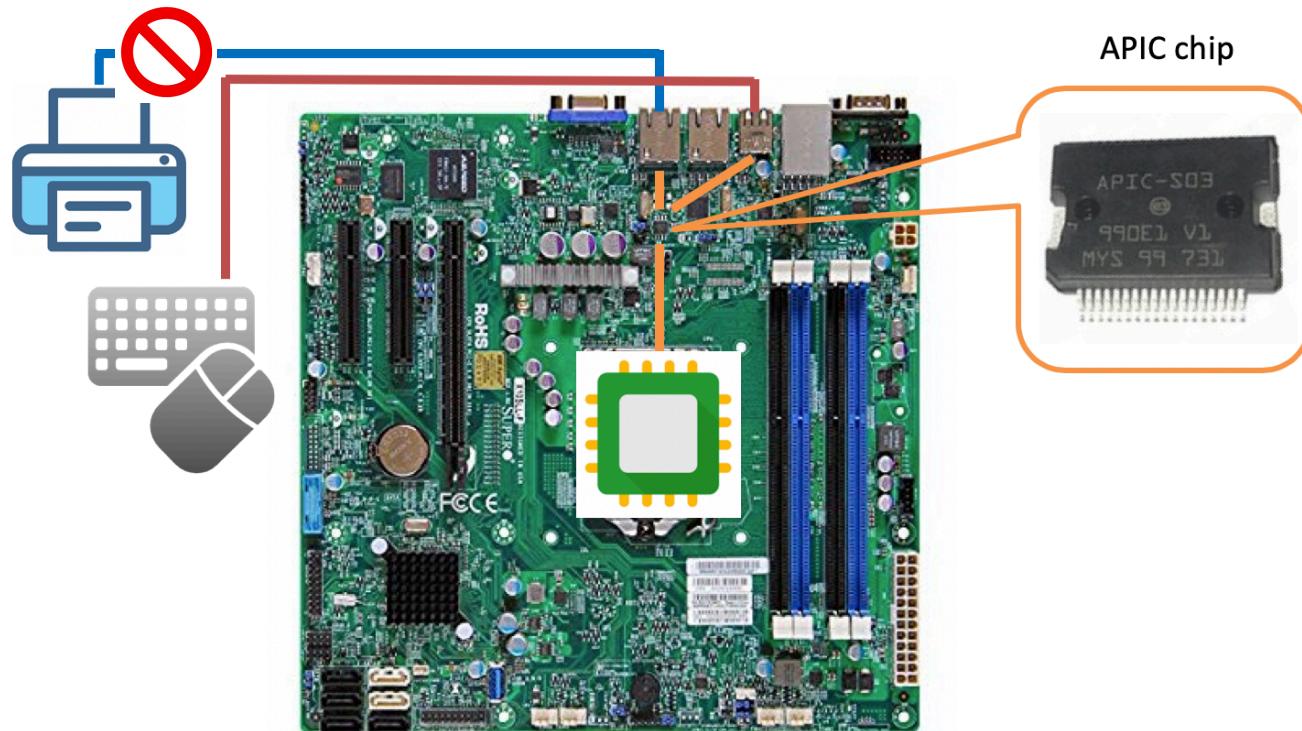
- APIC also needs to prioritize multiple requests

Mouse>Printer



APIC: Advanced programmable interrupt controller

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

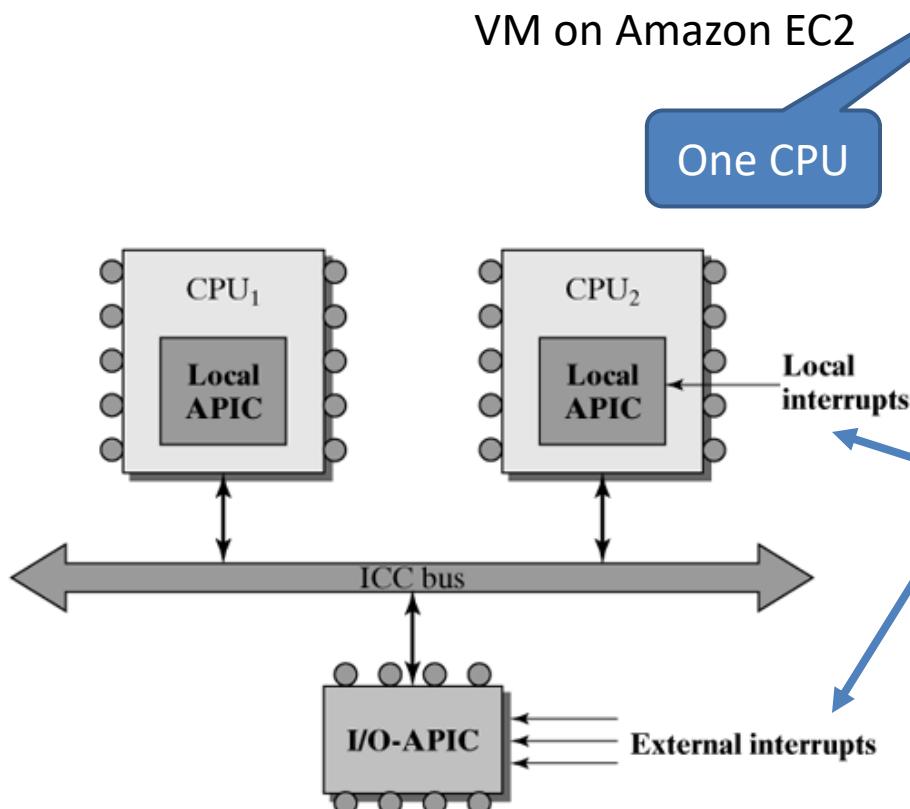


APIC: Advanced programmable interrupt controller

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



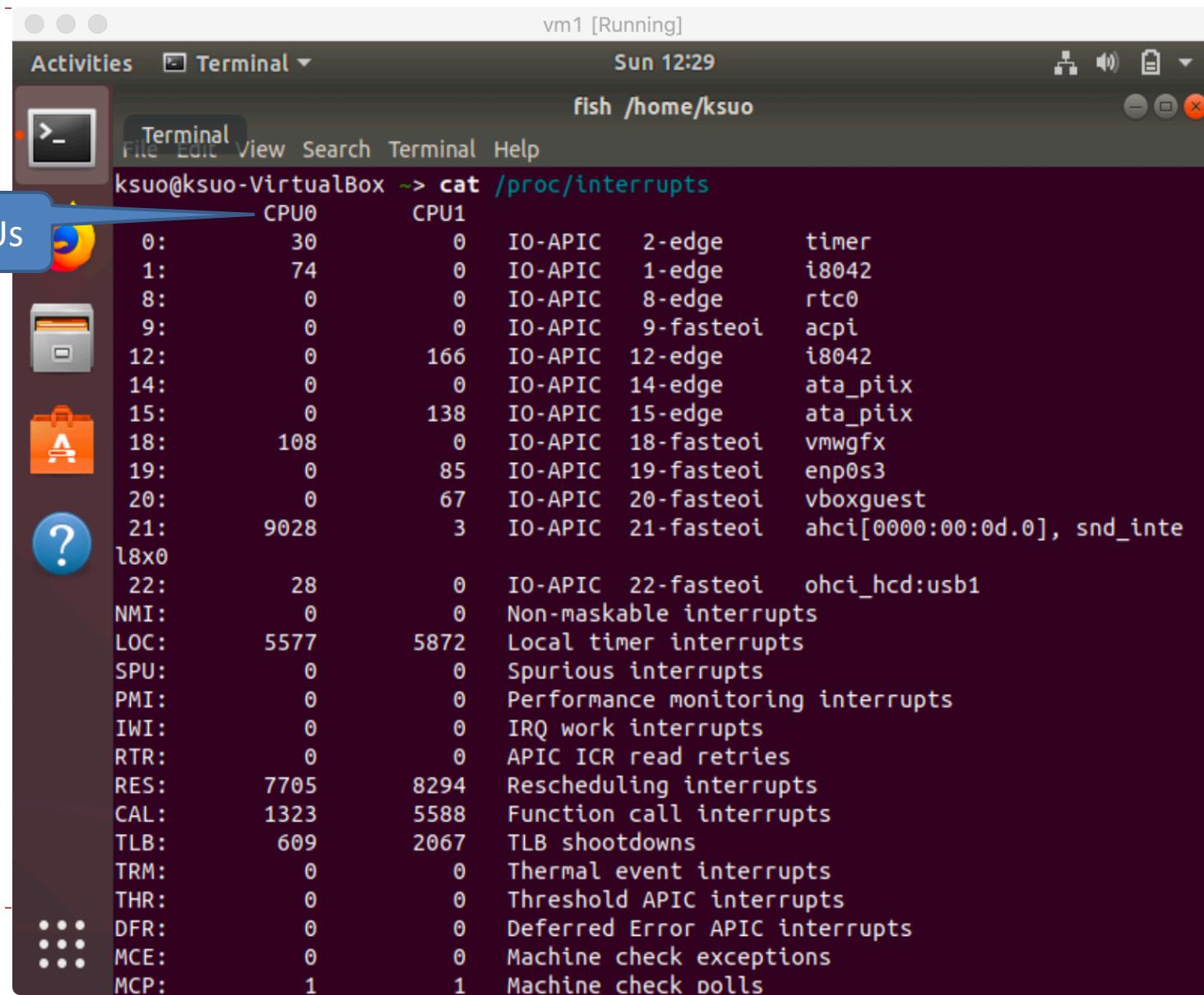
APIC: cat /proc/interrupts



```
1. ec2-user@ip-172-31-36-115:~ (ssh)
fish /home/pi/D... ● ⌘1 ec2-user@ip-172-31-36-115 ~] cat /proc/interrupts
[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
2:      2599  xen-pirq  4-ioapic-edge ttyS0
3:          2  xen-pirq  8-ioapic-edge rtc0
4:          0  xen-pirq  9-ioapic-level acpi
5:          0  xen-pirq 12-ioapic-edge i8042
6:          0  IO-APIC  14-edge      ata_piix
7:          0  IO-APIC  15-edge      ata_piix
8:        4827  xen-percpu -irq      timer0
9:          0  xen-percpu -ipi     resched0
10:         0  xen-percpu -ipi     callfunc0
11:         0  xen-percpu -irq     debug0
12:         0  xen-percpu -ipi     callfuncsingle0
13:         0  xen-percpu -ipi     spinlock0
14:         0  xen-dyn   -event    xenbus
15:      19103  xen-dyn   -event    eth0
16:       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



```
vm1 [Running]
Activities Terminal
Sun 12:29
fish /home/ksuo
Terminal File Edit View Search Terminal Help
ksuo@ksuo-VirtualBox ~> cat /proc/interrupts
CPU0      CPU1
0:        30      0  IO-APIC   2-edge    timer
1:        74      0  IO-APIC   1-edge    i8042
8:          0      0  IO-APIC   8-edge    rtc0
9:          0      0  IO-APIC   9-fasteoi acpi
12:         0     166  IO-APIC  12-edge    i8042
14:         0      0  IO-APIC  14-edge   ata_piix
15:         0     138  IO-APIC  15-edge   ata_piix
18:        108      0  IO-APIC 18-fasteoi vmwgfx
19:         0     85  IO-APIC 19-fasteoi enp0s3
20:         0     67  IO-APIC 20-fasteoi vboxguest
21:      9028      3  IO-APIC 21-fasteoi ahci[0000:00:0d.0], snd_inte
l8x0
22:        28      0  IO-APIC 22-fasteoi ohci_hcd:usb1
NMI:        0      0 Non-maskable interrupts
LOC:      5577    5872 Local timer interrupts
SPU:        0      0 Spurious interrupts
PMI:        0      0 Performance monitoring interrupts
IWI:        0      0 IRQ work interrupts
RTR:        0      0 APIC ICR read retries
RES:      7705    8294 Rescheduling interrupts
CAL:      1323    5588 Function call interrupts
TLB:       609    2067 TLB shootdowns
TRM:        0      0 Thermal event interrupts
THR:        0      0 Threshold APIC interrupts
DFR:        0      0 Deferred Error APIC interrupts
MCE:        0      0 Machine check exceptions
MCP:        1      1 Machine check polls
```

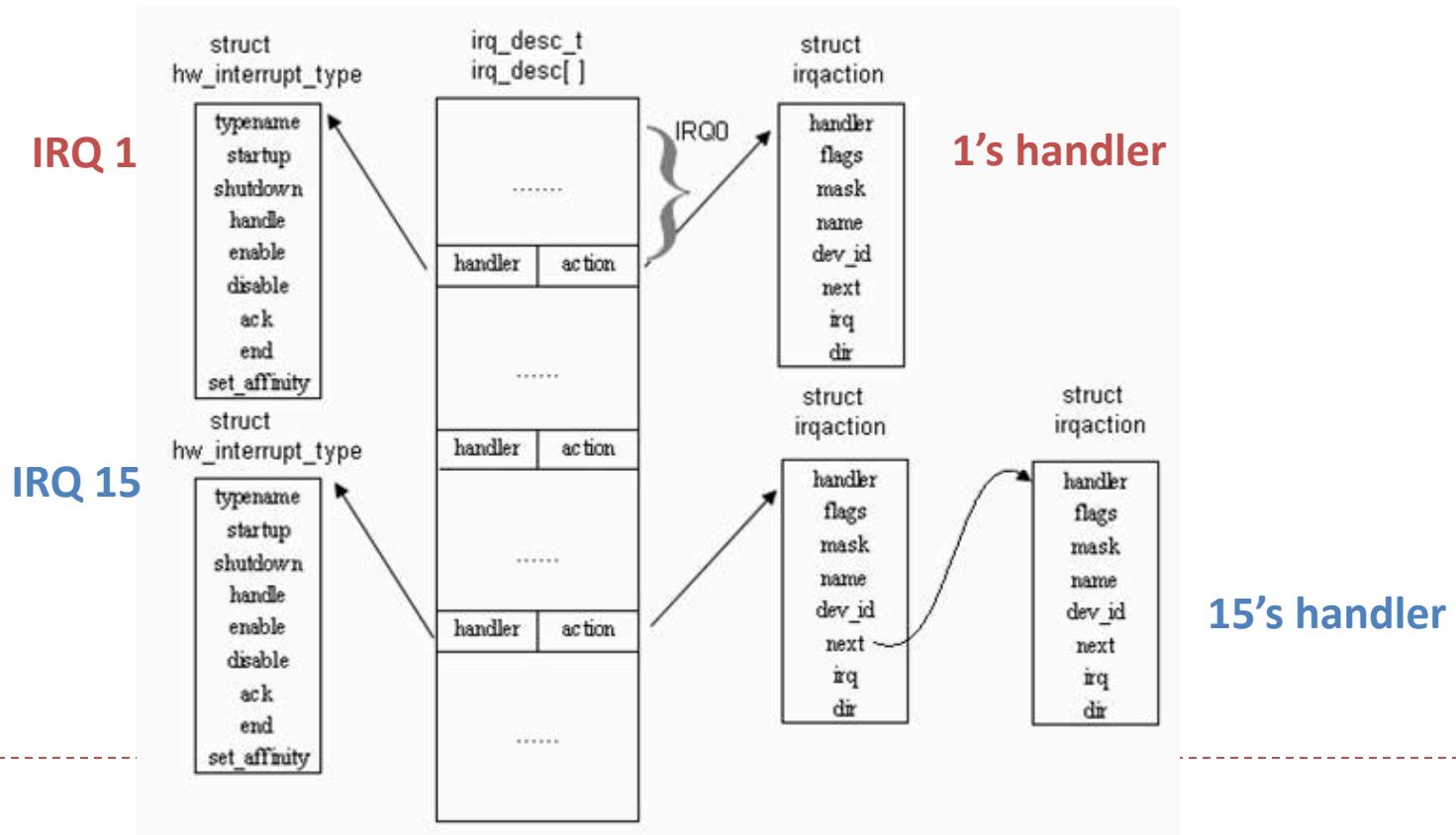
Outline

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



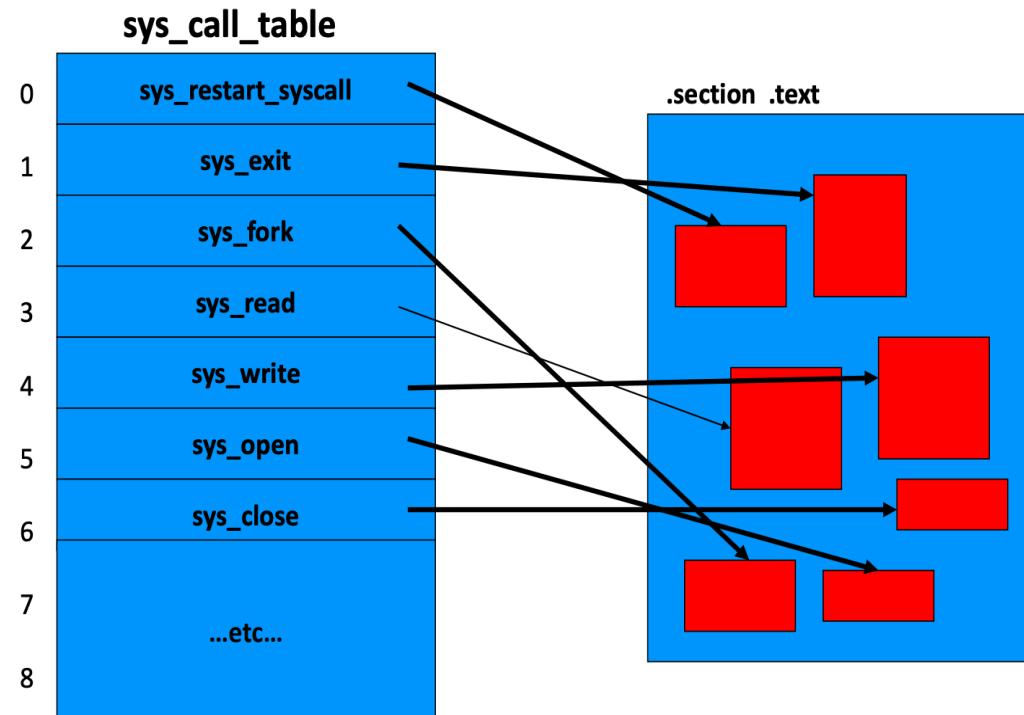
Interrupt Descriptor Table (IDT)

- 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



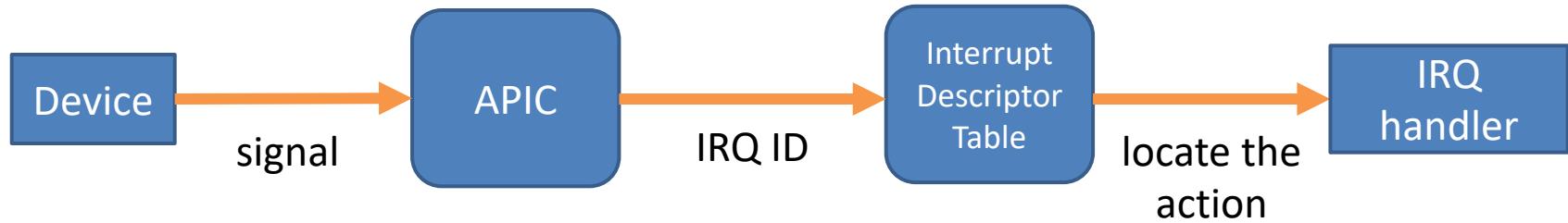
Similar as System call table

- 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

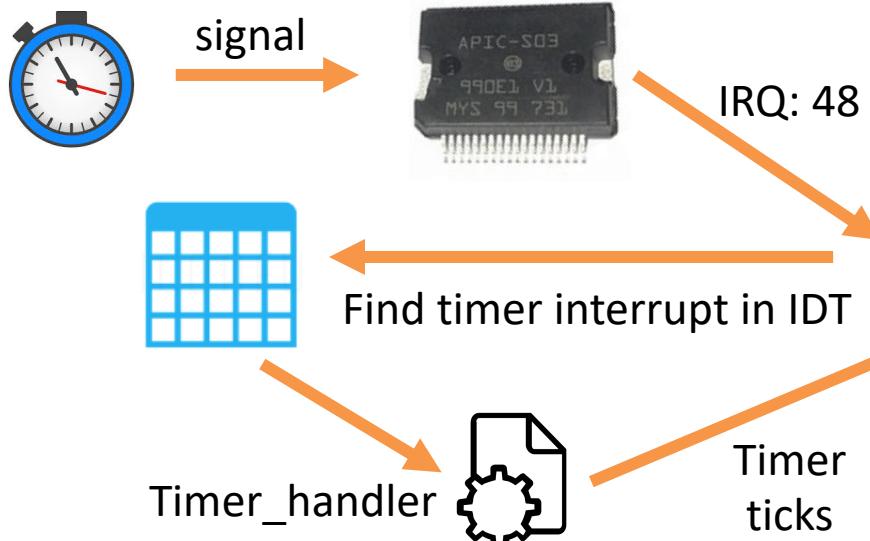


The ‘jump-table’ idea

The process of interrupt



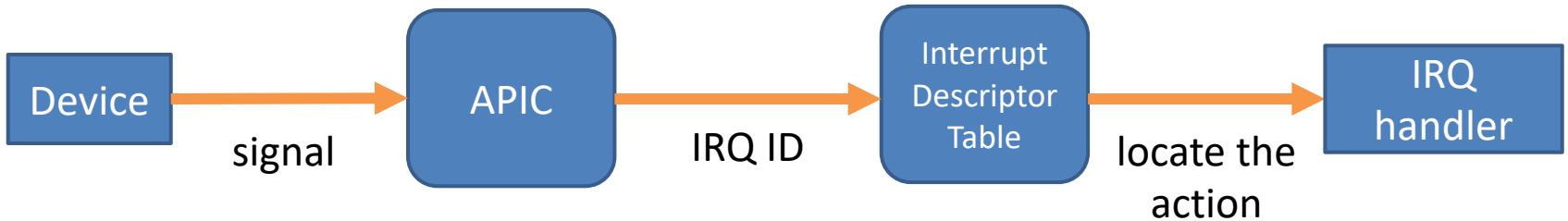
Take the *time interrupt* as an example



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	TO-APIC	15-edge	ata_piix
48:	43857	xen-percpu	-irq	timer0
49:	0	xen-percpu	-ipi	resched0
50:	0	xen-percpu	-ipi	callfunc0
51:	0	xen-percpu	-irq	debug0
52:	0	xen-percpu	-ipi	callfuncsingle0
53:	0	xen-percpu	-ipi	spinlock0
54:	254	xen-dyn	-event	xenbus



Interrupt handling



- When an interrupt is triggered (interrupt request, or IRQ), the processor:
 - 1. enters privileged mode,
 - 2. records state necessary to process the interrupt,
 - 3. jumps to a pre determined memory location and begins executing instructions.

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



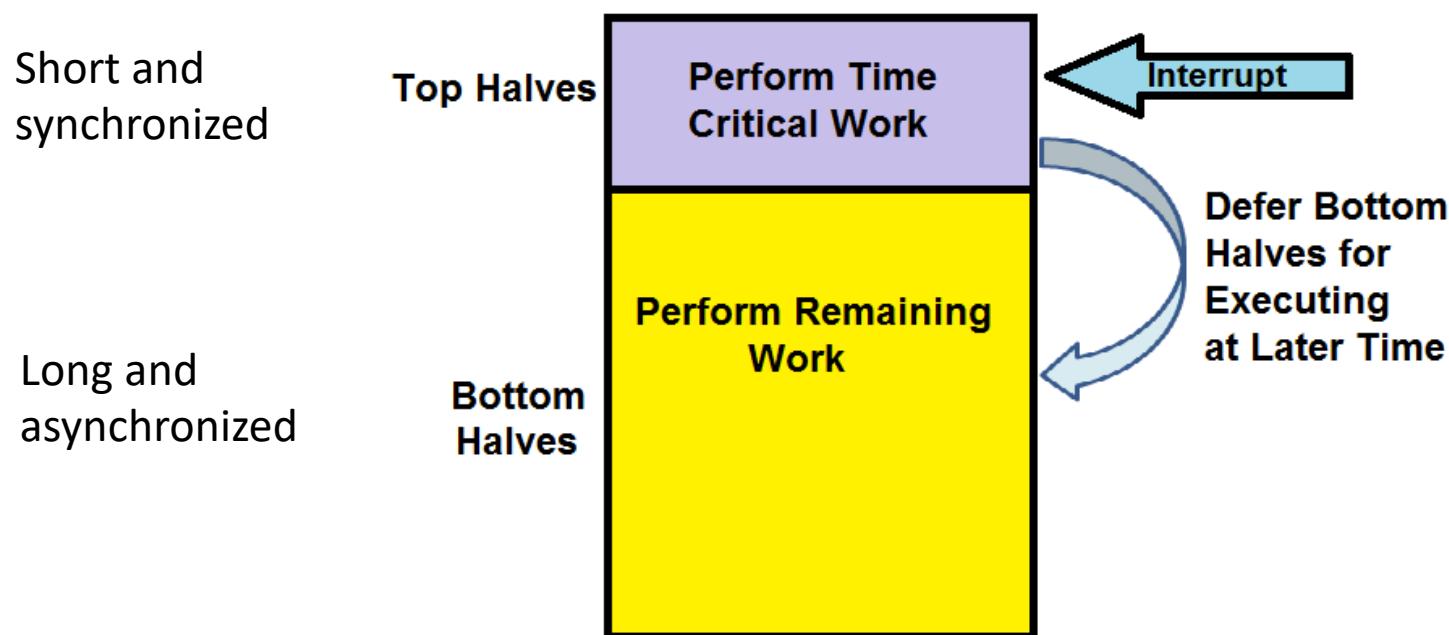
Top-half and bottom-half in interrupts

- 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
- Processors implement multiple interrupt lines, input wires on which a logic transition (or level) will trigger an interrupt.



Hardware interrupt: cat /proc/interrupts

```
pi@raspberrypi ~/Downloads> cat /proc/interrupts
CPU0      CPU1      CPU2      CPU3
16:        0        0        0        0  bcm2836-timer    0 Edge    arch_timer
17:  21666    63962    23766    9395  bcm2836-timer    1 Edge    arch_timer
21:        0        0        0        0  bcm2836-pmu     9 Edge    arm-pmu
23:    3600      0        0        0  ARMCTRL-level   1 Edge
24:      29      0        0        0  ARMCTRL-level   2 Edge    VCHIQ doorbell
46:        0        0        0        0  ARMCTRL-level  48 Edge    bcm2708_fb dma
48:        0        0        0        0  ARMCTRL-level  50 Edge    DMA IRQ
50:        0        0        0        0  ARMCTRL-level  52 Edge    DMA IRQ
51:    360      0        0        0  ARMCTRL-level  53 Edge    DMA IRQ
54:   4908      0        0        0  ARMCTRL-level  56 Edge    DMA IRQ
59:        0        0        0        0  ARMCTRL-level  61 Edge    bcm2835-auxirq
62: 139344      0        0        0  ARMCTRL-level  64 Edge    dwc_otg, dwc_otg_pcd, dwc_otg_hcd:usb1
86:    870      0        0        0  ARMCTRL-level  88 Edge    mmc0
87:   7237      0        0        0  ARMCTRL-level  89 Edge    uart-pl011
92: 101080      0        0        0  ARMCTRL-level  94 Edge    mmc1
169:        0        0        0        0  lan78xx-irqs  17 Edge    usb-001:004:01
FIQ:      usb_fiq
IPI0:        0        0        0        0  CPU wakeup interrupts
IPI1:        0        0        0        0  Timer broadcast interrupts
IPI2:    7855    81594    13799    6452  Rescheduling interrupts
IPI3:      10       6       9       8  Function call interrupts
IPI4:        0        0        0        0  CPU stop interrupts
IPI5:    7342    26013    5157    2072  IRQ work interrupts
IPI6:        0        0        0        0  completion interrupts
Err:        0        0        0        0
```

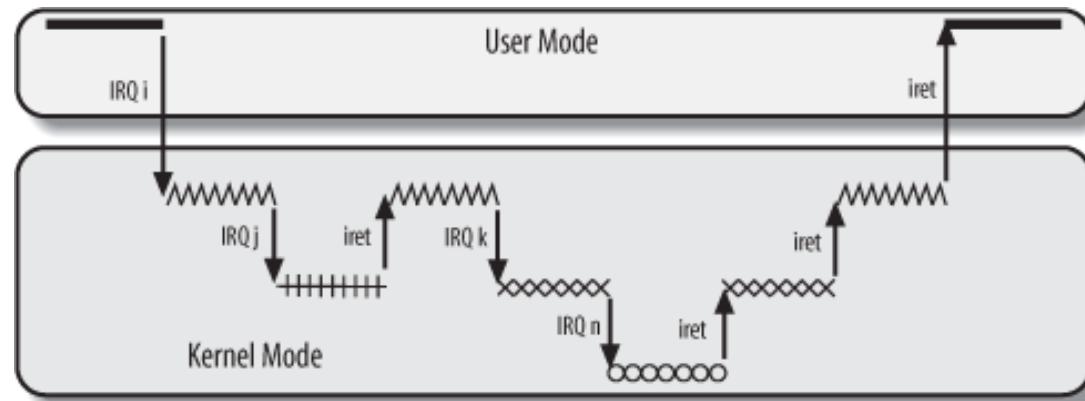


\$ watch -n 1 -d cat /proc/interrupts

```
watch /home/ksuo
Every 1.0s: cat /proc/interrupts
LinuxKernel2: Sun Feb 16 23:21:05 2020
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  143121561  26274980 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          0         0         0
MIS:      0          0         0         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
```

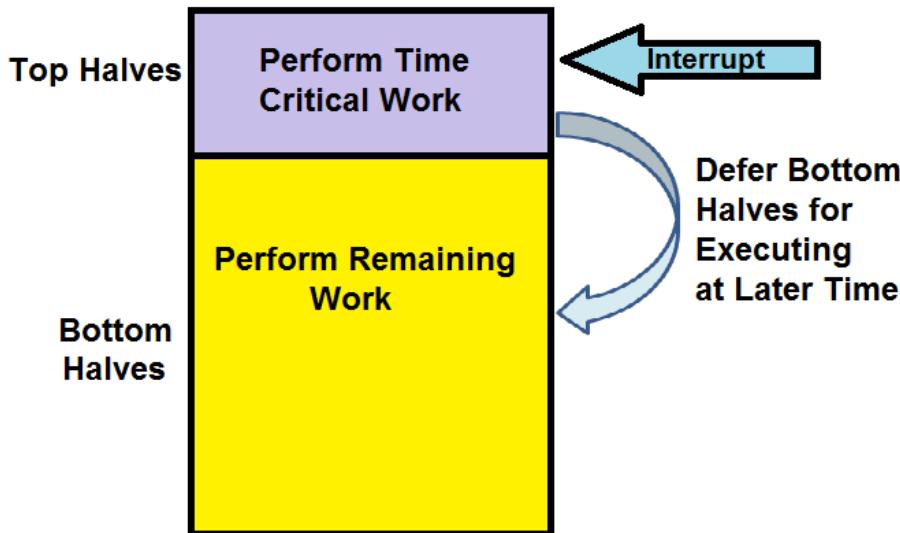
Hardware interrupt feature

- **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

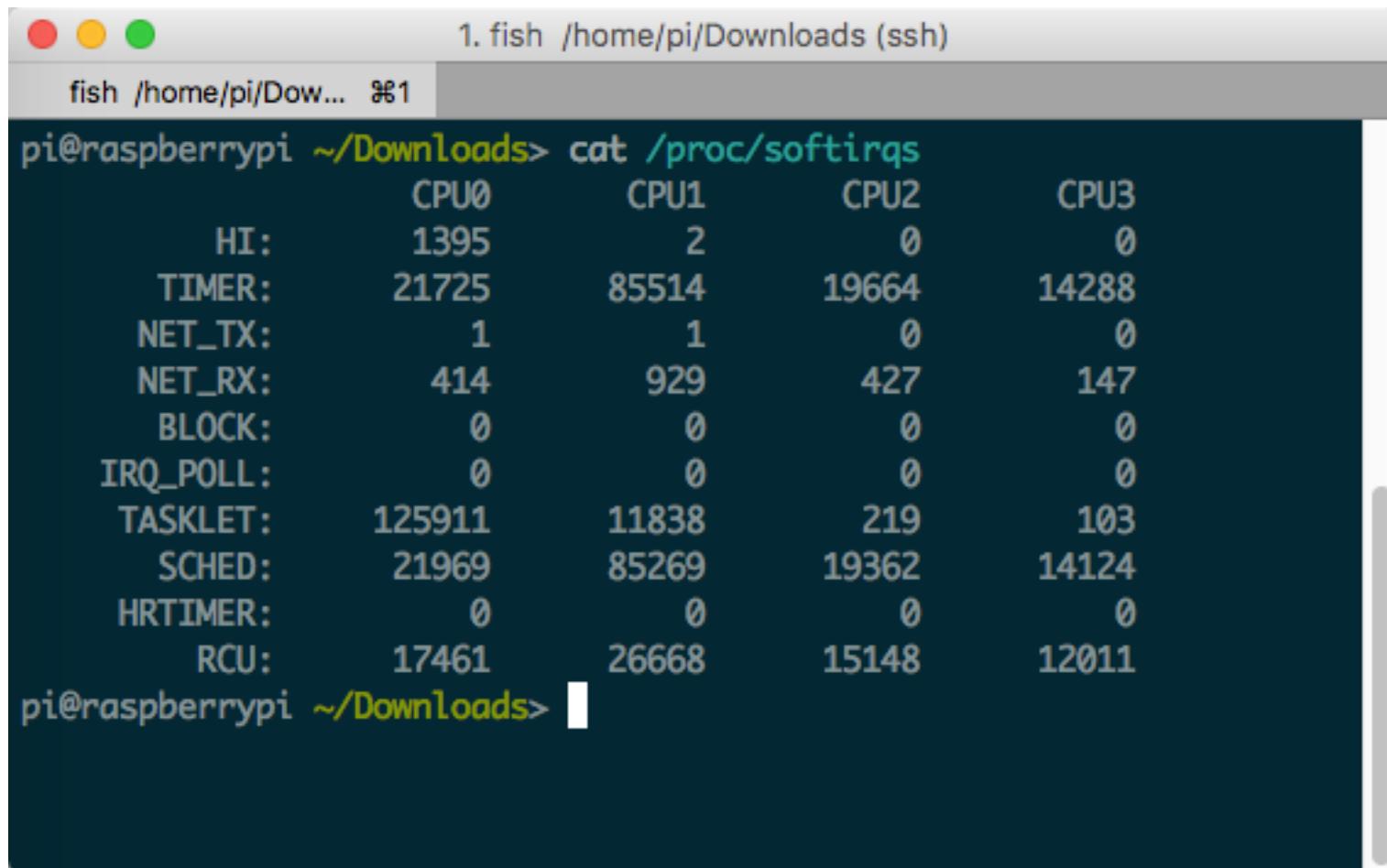


Software interrupt feature

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



A screenshot of a terminal window titled "fish /home/pi/Downloads (ssh)". The command "cat /proc/softirqs" is run, displaying a table of software interrupt counts across four CPUs (CPU0, CPU1, CPU2, CPU3). The table shows various interrupt types and their counts, such as TIMER, NET_RX, and RCU.

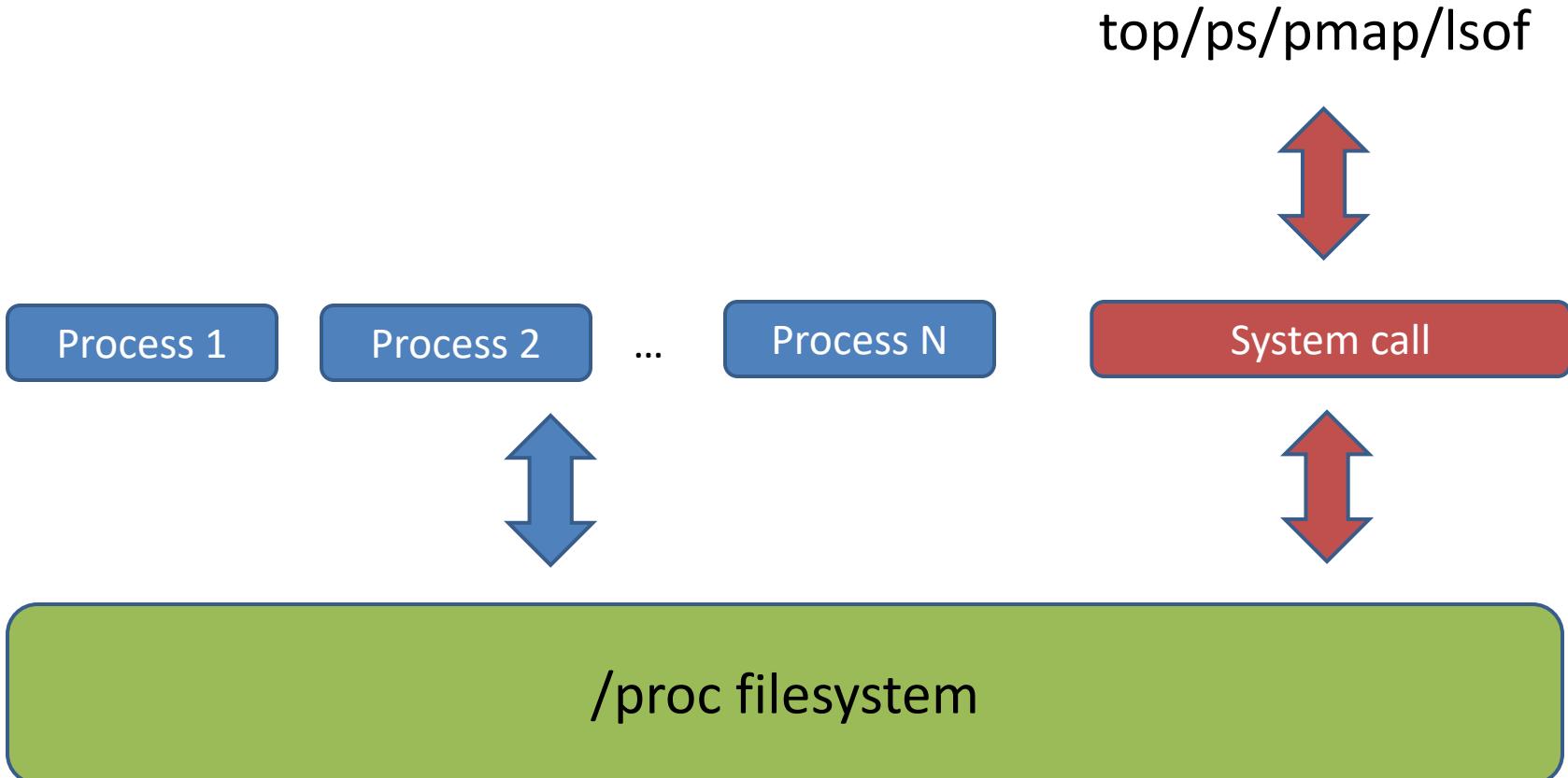
	CPU0	CPU1	CPU2	CPU3
HI:	1395	2	0	0
TIMER:	21725	85514	19664	14288
NET_TX:	1	1	0	0
NET_RX:	414	929	427	147
BLOCK:	0	0	0	0
IRQ_POLL:	0	0	0	0
TASKLET:	125911	11838	219	103
SCHED:	21969	85269	19362	14124
HRTIMER:	0	0	0	0
RCU:	17461	26668	15148	12011

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

```
watch /home/ksuo
Every 1.0s: cat /proc/softirqs                               LinuxKernel2: Sun Feb 16 23:26:54 2020

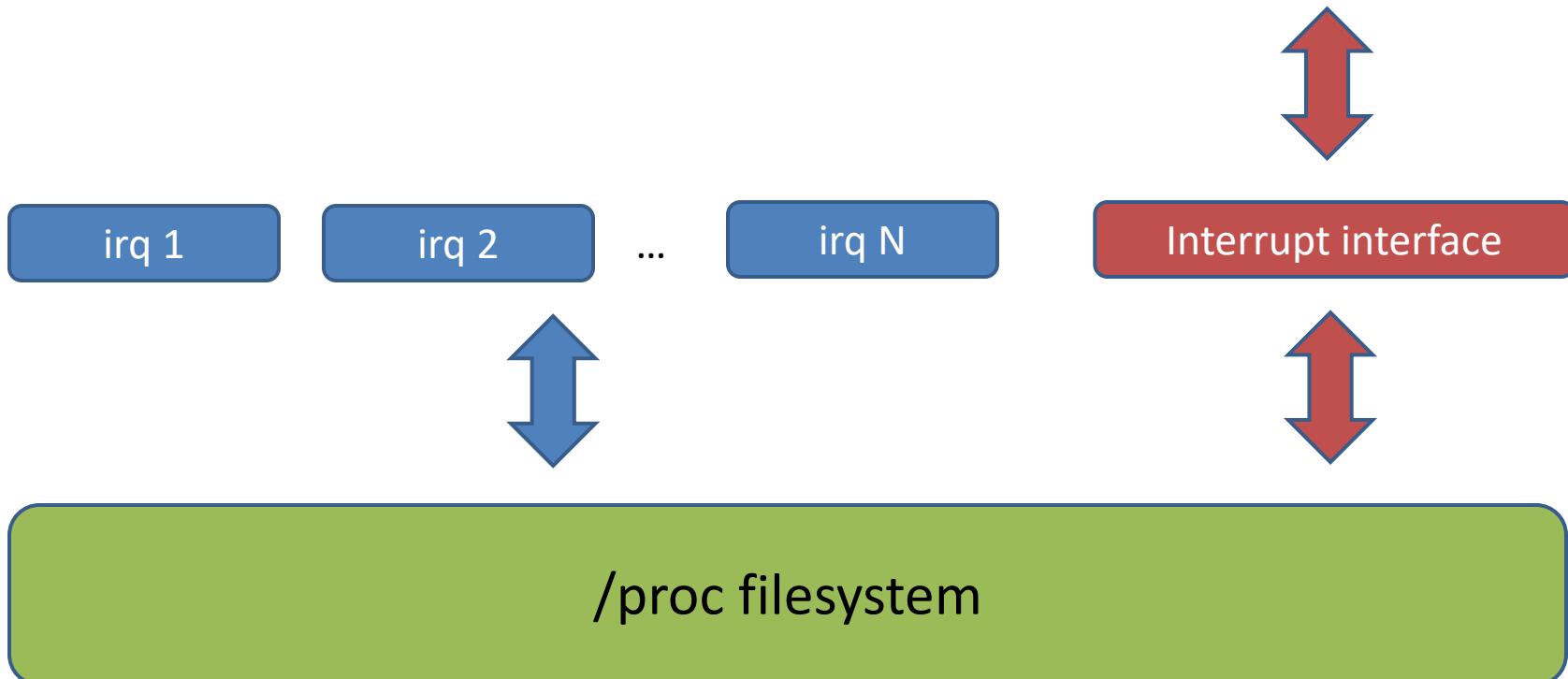
          CPU0       CPU1       CPU2       CPU3
HI:          0         1         0         0
TIMER:    11005104    7993553    75625648    11401153
NET_TX:        0         1         8         1
NET_RX:     1211       703       778      10282130
BLOCK:        0         0         0         0
IRQ_POLL:      0         0         0         0
TASKLET:      14         2         0         0
SCHED:     9317734    7430993    73278709    10394289
HRTIMER:       0         0         0         0
RCU:      7450009    2604225    33874861    4333671
```

System call: the `/proc` filesystem



Interrupt: the `/proc` filesystem

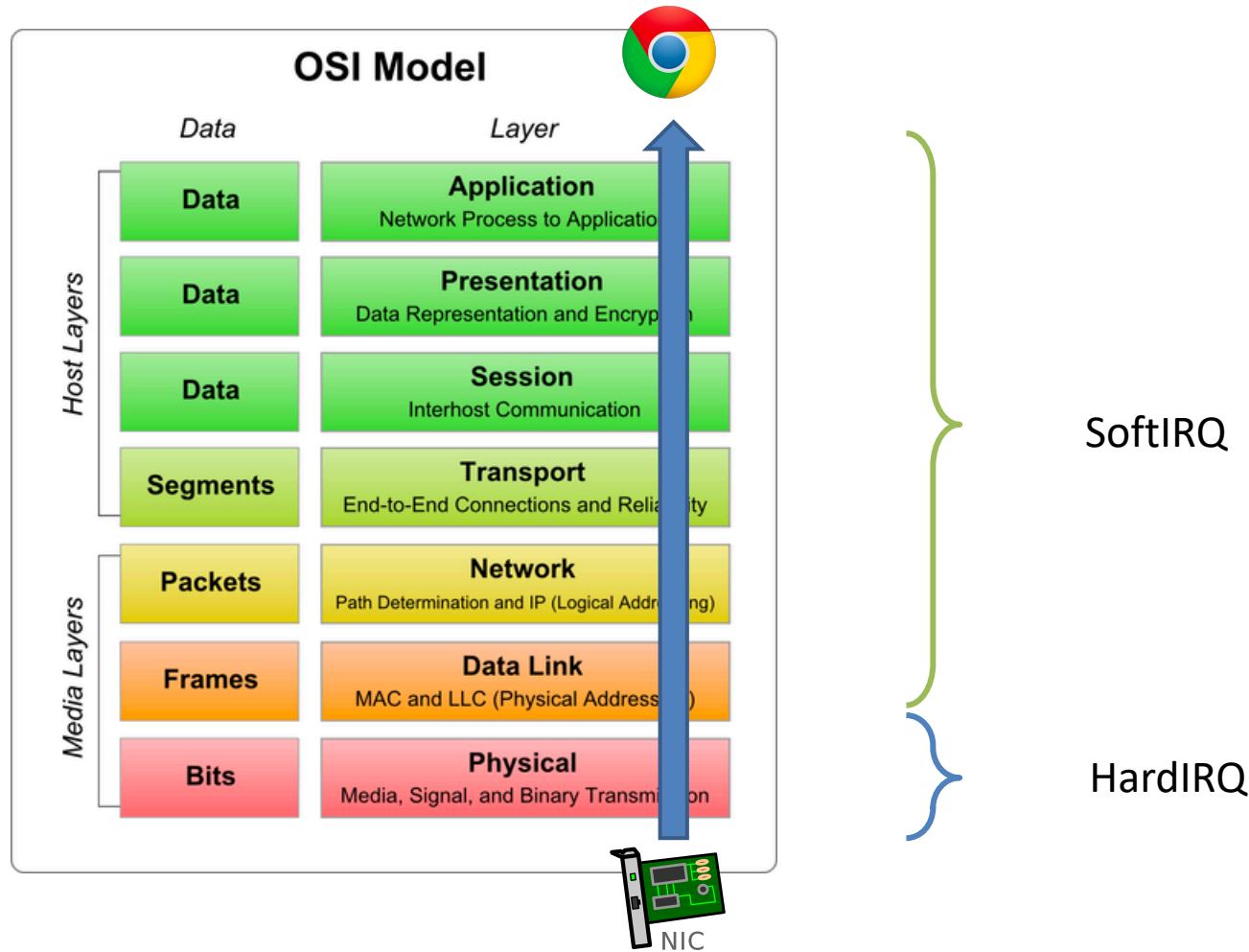
`cat /proc/interrupts`
`cat /proc/softirqs`



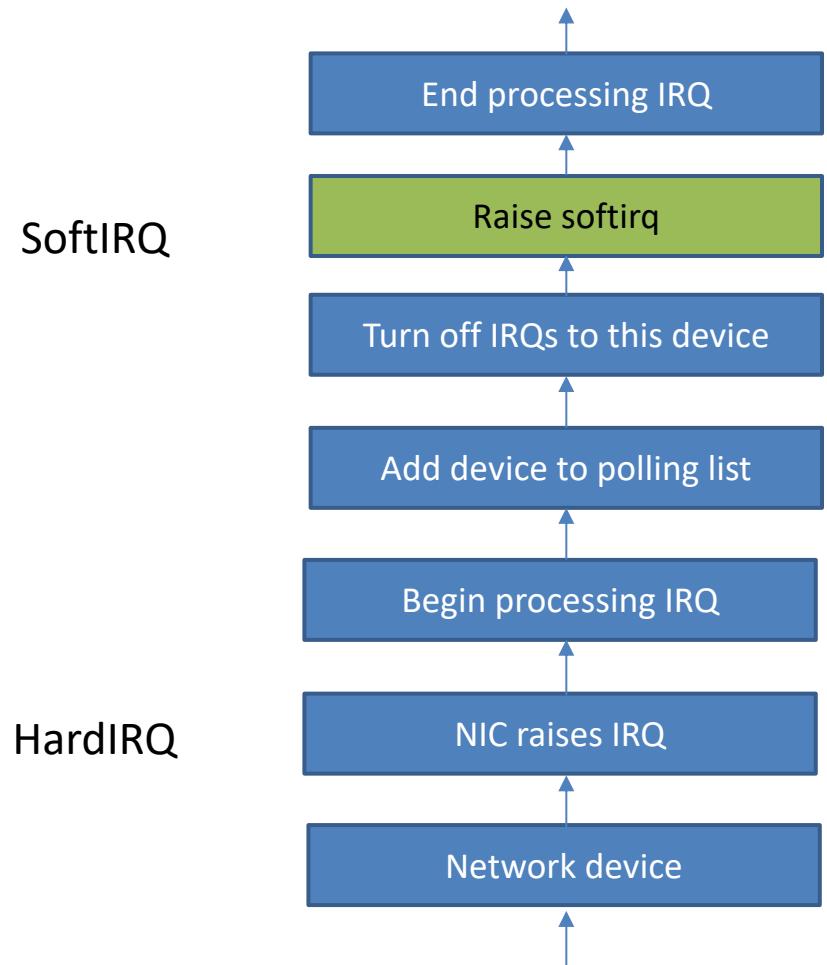
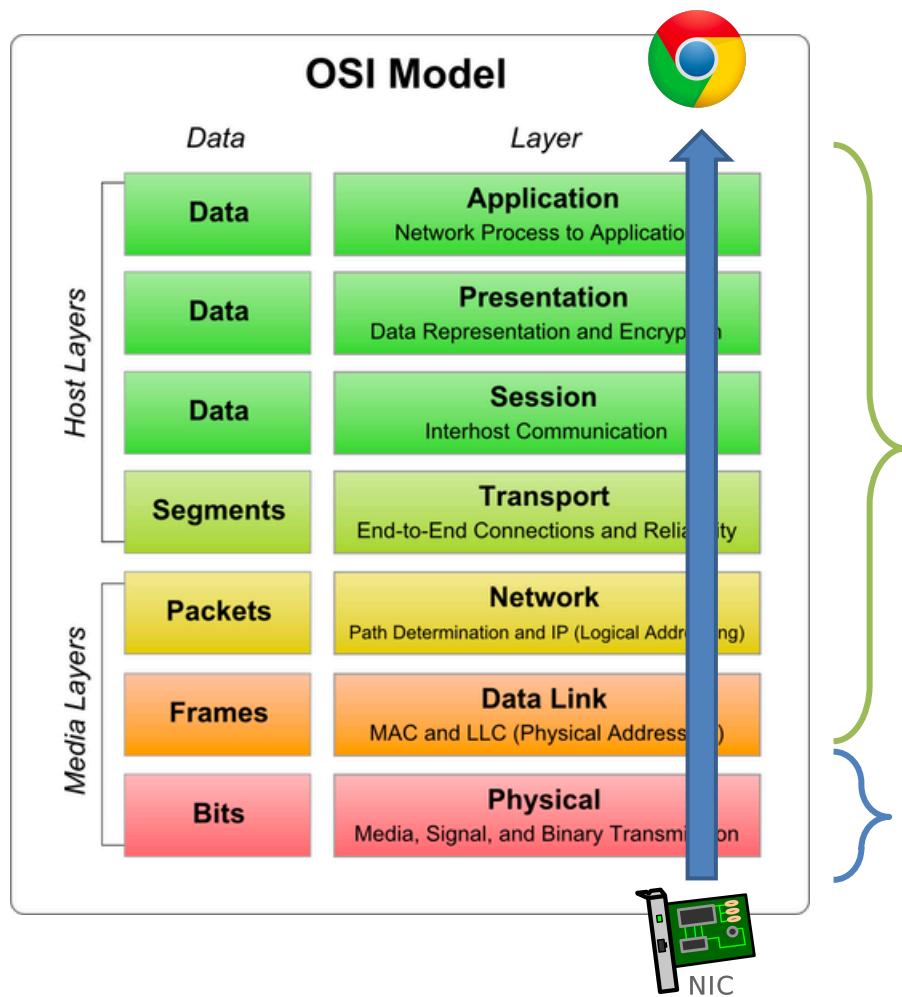
Hardware IRQ vs. Software IRQ

	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 example: Network interrupt



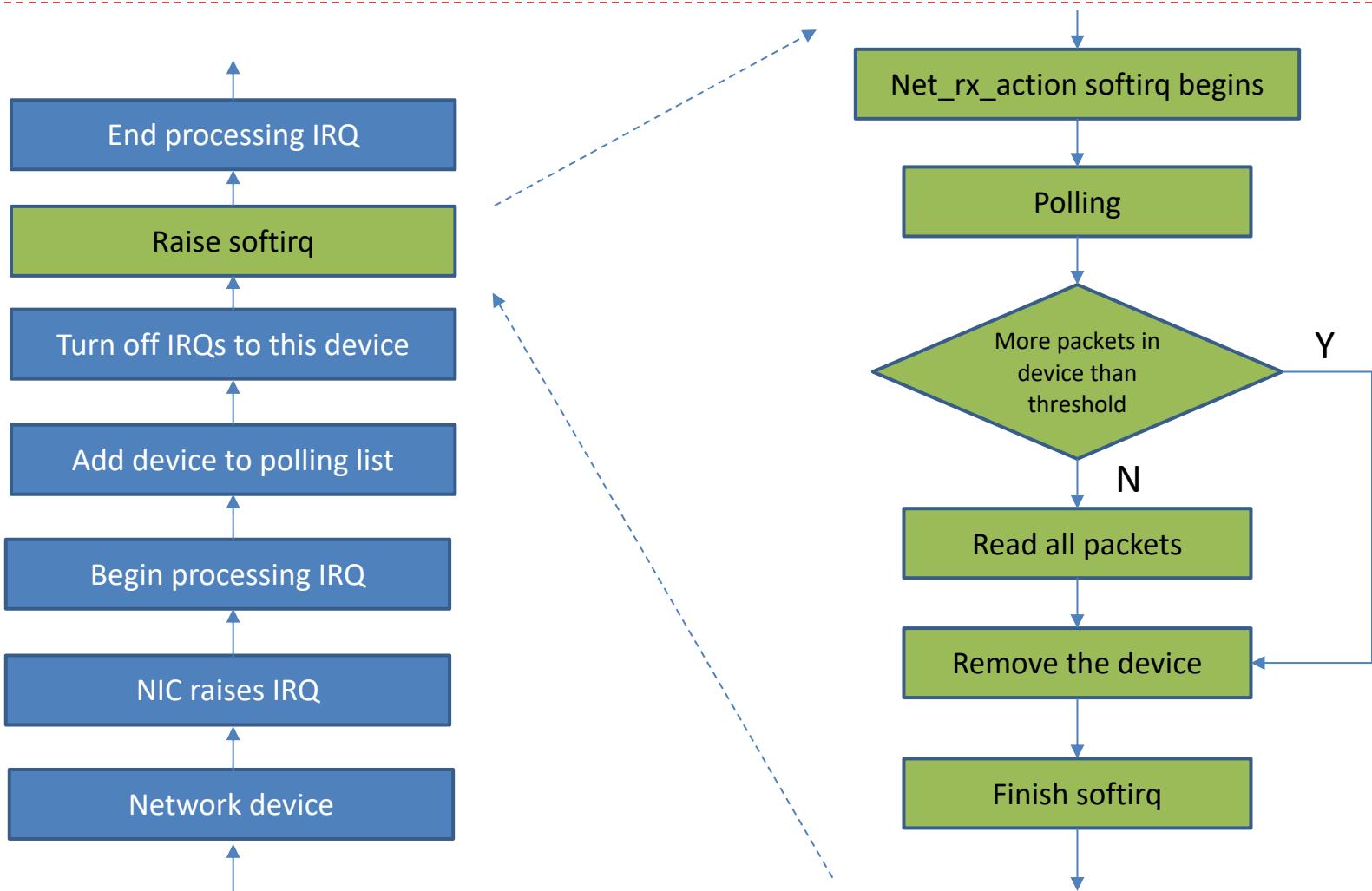
Interrupt example: Network interrupt



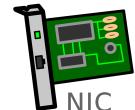
SoftIRQ

HardIRQ

Interrupt example: Network interrupt

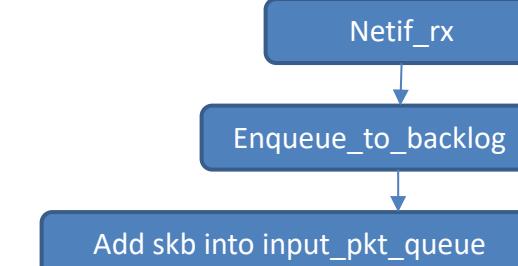


Interrupt example: Network interrupt



Top half
Hardware
IRQ

<https://elixir.bootlin.com/linux/latest/source/net/core/dev.c#L4544>

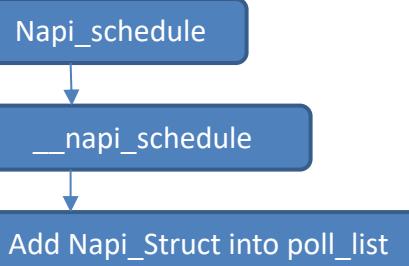


<https://elixir.bootlin.com/linux/latest/source/net/core/dev.c#L6399>

<https://elixir.bootlin.com/linux/latest/source/net/core/dev.c#L5913>

Bottom half
Software
IRQ

<https://elixir.bootlin.com/linux/latest/source/include/linux/netdevice.h#L438>

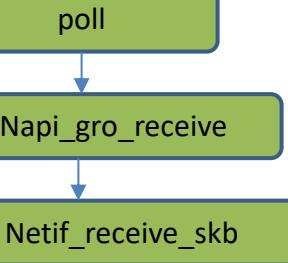


SoftIRQ

Net_rx_action

Process_backlog

__netif_receive_skb

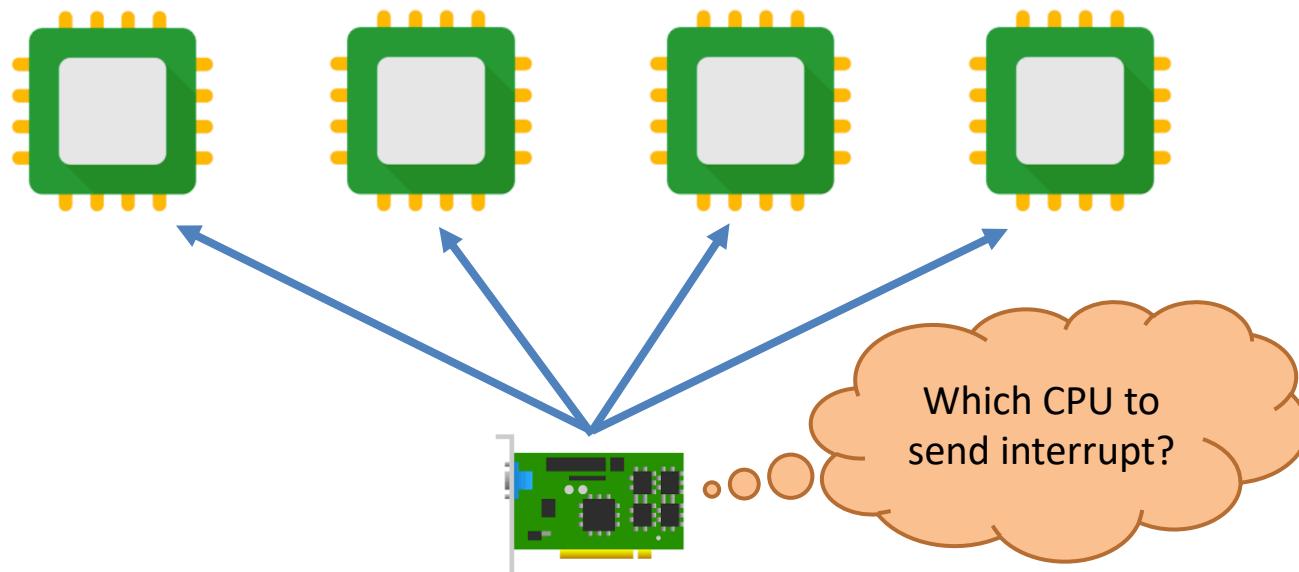


<https://elixir.bootlin.com/linux/latest/source/net/core/dev.c#L5265>



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

```
fish /home/pi/Dow... %1
1. fish /home/pi/Downloads (ssh)
fish /home/pi/Downloads> cat /proc/interrupts
CPU0      CPU1      CPU2      CPU3
16:        0         0         0   bcm2836-timer    0 Edge    arch_timer
17:  21666    63962    23766    9395  bcm2836-timer    1 Edge    arch_timer
21:        0         0         0   bcm2836-pmu     9 Edge    arm-pmu
23:    3600      0         0       0 ARMCTRL-level  1 Edge    3f00b880.mailbox
24:      29      0         0       0 ARMCTRL-level  2 Edge    VCHIQ doorbell
46:        0         0         0       0 ARMCTRL-level 48 Edge   bcm2708_fb dma
48:        0         0         0       0 ARMCTRL-level 50 Edge   DMA IRQ
50:        0         0         0       0 ARMCTRL-level 52 Edge   DMA IRQ
51:    360      0         0       0 ARMCTRL-level 53 Edge   DMA IRQ
54:    4908      0         0       0 ARMCTRL-level 56 Edge   DMA IRQ
59:        0         0         0       0 ARMCTRL-level 61 Edge   bcm2835-auxirq
62:  139344      0         0       0 ARMCTRL-level 64 Edge   dwc_otg, dwc_otg_pcd, dwc_otg_hcd:usb1
86:     870      0         0       0 ARMCTRL-level 88 Edge   mmc0
87:    7237      0         0       0 ARMCTRL-level 89 Edge   uart-pl011
92:  101080      0         0       0 ARMCTRL-level 94 Edge   mmc1
169:        0         0         0       0 lan78xx-irqs 17 Edge   usb-001:004:01
FIQ:      sb_fiq
IPI0:        0         0         0       0 CPU wakeup interrupts
IPI1:        0         0         0       0 Timer broadcast interrupts
IPI2:    7855    81594    13799    6452 Rescheduling interrupts
IPI3:      10       6       9       8 Function call interrupts
IPI4:        0         0         0       0 CPU stop interrupts
IPI5:    7342   26013    5157    2072 IRQ work interrupts
IPI6:        0         0         0       0 completion interrupts
Err:        0         0         0         0
```

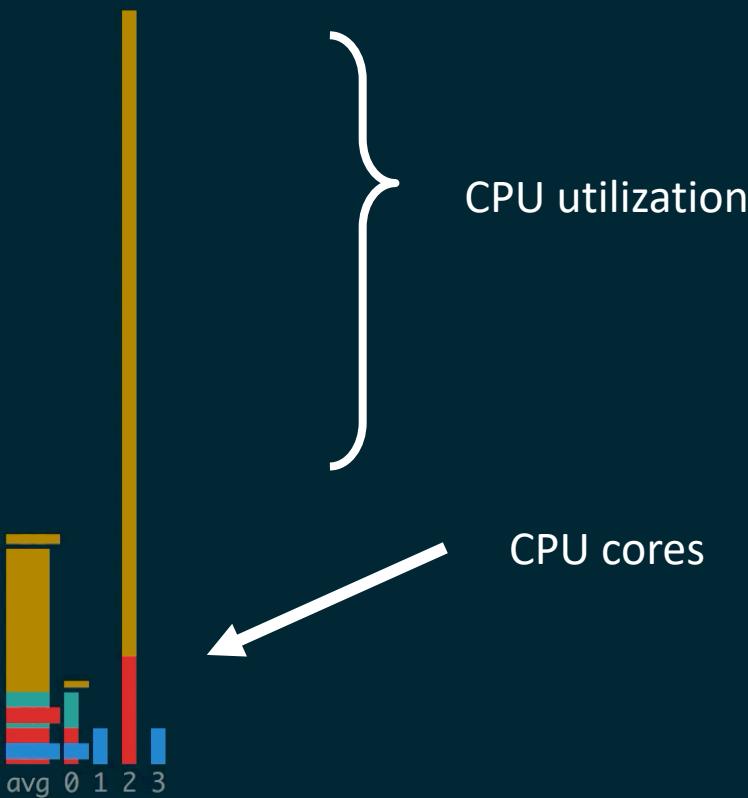
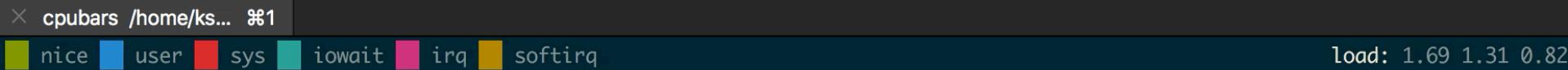


1. nmon /home/ksuo (ssh)

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

CPU Utilisation Stats											
ALL	13.0	0.0	17.0	271.4	16.0	0.0	77.0	0.0	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)



Set Interrupt Affinity to certain CPU cores

CPU utilization

CPU cores

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

