
Interrupts and System Calls

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

(Based on slides by Don Porter at UNC)

<https://www.cs.unc.edu/~porter/>



Questions from last session

```
section .data ;Data segment
    userMsg db 'Please enter a number: ' ;Ask the user to enter a number
    lenUserMsg equ $-userMsg ;The length of the message
    dispMsg db 'You have entered: '
    lenDispMsg equ $-dispMsg

section .bss ;Uninitialized data
    num resb 5

section .text ;Code Segment
    global _start

_start: ;User prompt
    mov eax, 4
    mov ebx, 1
    mov ecx, userMsg
    mov edx, lenUserMsg
    int 80h
```

دایرکٹ کال پسند

db: data byte

equ: equivalent

resb: reserve byte

https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm



Questions from last session

زبان های برنامه نویسی

Compiler vs interpreter

java - python

بعد مرئی رفعی رجوای → اپلیکیشن python را تویی
OS موردنظر شنید لذت
من توانم اجرای آن
تویی هر جایی.

* java is considered both compiler & interpreter.

int instruction

– The instruction generates a **software call** to an interrupt handler.

– Example

• Trap to debugger: int \$3

• Trap to interrupt 0xff: int \$0xff

<https://docs.oracle.com/cd/E19455-01/806-3773/instructionset-74/index.html>



Background: Control Flow

```
x = 2, y = true
if (y) {
    2 != x;
    printf(x);
} //...
```

```
void printf(va_args)
{
    //...
}
```



Background: Control Flow

pc

```
x = 2, y = true      void printf(va_args)
if (y) {
    2 /= x;           //
    printf(x);         }
} //...
```



Background: Control Flow

pc

```
x = 2, y = true      void printf(va_args)
if (y) {
    2 /= x;           {
                      //...
    printf(x);        }
} //...
```



Background: Control Flow

```
x = 2, y = true      void printf(va_args)
if (y) {                {
    2 /= x;          // ...
    printf(x);        }
} // ...
```

pc



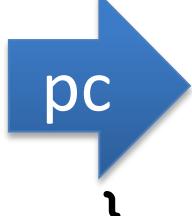
Background: Control Flow

```
x = 2, y = tr pc → void printf(va_args)
if (y) {
    2 /= x;           // ...
    printf(x);        }
} // ...
```



Background: Control Flow

```
x = 2, y = true      void printf(va_args)
if (y) {
    2 /= x;
    printf(x);
} //...
```



Background: Control Flow

```
x = 2, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
}  
} //...
```

pc

```
void printf(va_args)  
{  
    // ...  
}
```

يعنى خطبة خط اجرامى كدة.

• ... تابع for if فى

Regular control flow: branches and calls
(logically follows source code)



Background: Control Flow

```
x = 0, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
} //...
```

Background: Control Flow

pc → **x = 0, y = true**
if (y) {
 2 /= x;
 printf(x);
} //...

Background: Control Flow

```
x = 0, y = true
pc if (y) {
    2 /= x;
    printf(x);
} //...
```

Background: Control Flow

```
x = 0, y = true
if (y) {
    2 /= x;
    printf(x);
} //...
```

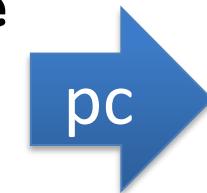
pc →

Divide by zero!
Program can't make progress!

~~2
0~~!

Background: Control Flow

```
x = 0, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
} //...
```



void

handle_divzero () {

x = 2;

}

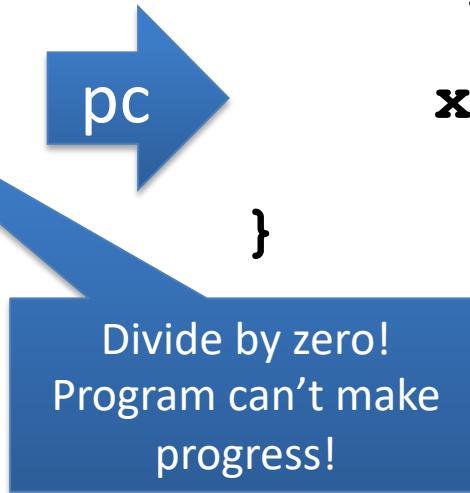
Divide by zero!
Program can't make
progress!

.exception handler

Background: Control Flow

```
x = 0, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
} //...
```

```
void  
handle_divzero() {  
    x = 2;  
}
```



Background: Control Flow

```
x = 0, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
} // ...
```

pc

```
void  
handle_divzero() {  
    x = 2;  
}
```

Divide by zero!
Program can't make
progress!



Background: Control Flow

```
x = 0, y = true  
if (y) {  
    2 /= x;  
    printf(x);  
}  
pc } //...
```

```
void  
handle_divzero() {  
    x = 2;  
}  
}
```

Divide by zero!
Program can't make
progress!

null exception, ...

int x = NULL;
printf(n); X null exception

Irregular control flow: exceptions, system calls, etc.

سُنْتَ بِنْ سُنْدُمْ
آقا تارَانْ طَافَ حَوَاسِطَ أَهْلَكَ مَا سُنْتَ
جُونْ مُنْظَمْ بَنْدُو دَيْلَ حَذَّرَ حَذَّرَ نَوْنَسْتَ سَقْمَ إِجْرَانْ

اين جينجىن دىرىتىلىنىڭ بى تىپىرىنىڭ فەرقىدا



Lecture goal

- Understand the hardware tools available for **irregular control flow**.
 - I.e., things other than a branch in a running program
- Building blocks for context switching, device management, etc.

Two types of interrupts

صونان با اجری یافتن

- **Synchronous:** will happen **every time** an instruction executes **(with a given program state)**
 - Divide by zero
 - System call → **read** (لرید زدن)
 - Bad pointer dereference → **null** (نول) **چیز** **با**
- **Asynchronous:** caused by an **external event**
 - Usually device I/O
 - Timer ticks (well, clocks can be considered a device)



Intel nomenclature

نام‌گذاری

- **Interrupt** – only refers to **asynchronous** interrupts
(...، IO، device event، ...)
(سخت افزاری)
- **Exception** – **synchronous** **control transfer**
(trap to OS)
trap event
لترنل دایرکت کنترل های دیگر.
* بازگشایی (سترن)
* توی func هر قدر بگرد بازگشت دست پاس.
وی از لحظه ای آن exception دیگر رخ نمی‌نماید.
OS می‌بیند لترنل می‌باشد.
- Note: from the programmer's perspective, these are handled with the **same abstractions**



Interrupt overview

- Each interrupt or exception includes a number indicating its type.
- E.g., 14 is a page fault, 3 is a debug breakpoint.
↳ .interrupt نماینده اینکه breakpoint
- This number is the index into an interrupt table.

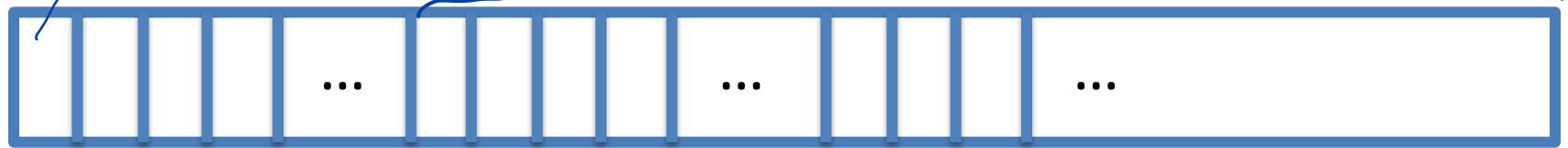


x86 interrupt table

interrupt ای نیز آدرسی است
روکی باید هنگام کردن.



لکی هر کدام از این خونهای آدرسی



0

31

47

255

Reserved for
the CPU

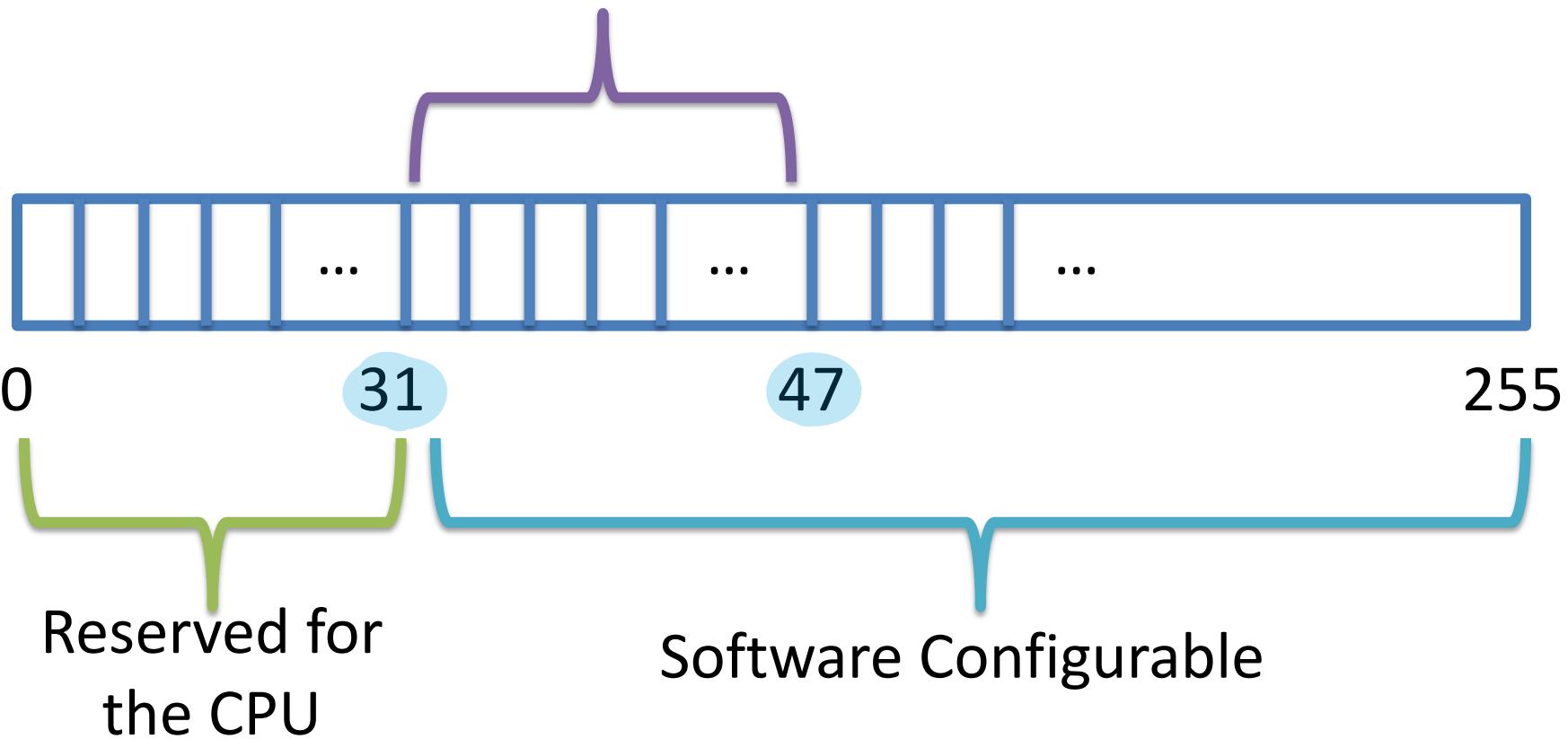
Software Configurable

(خودم می توانم فرآوریم).

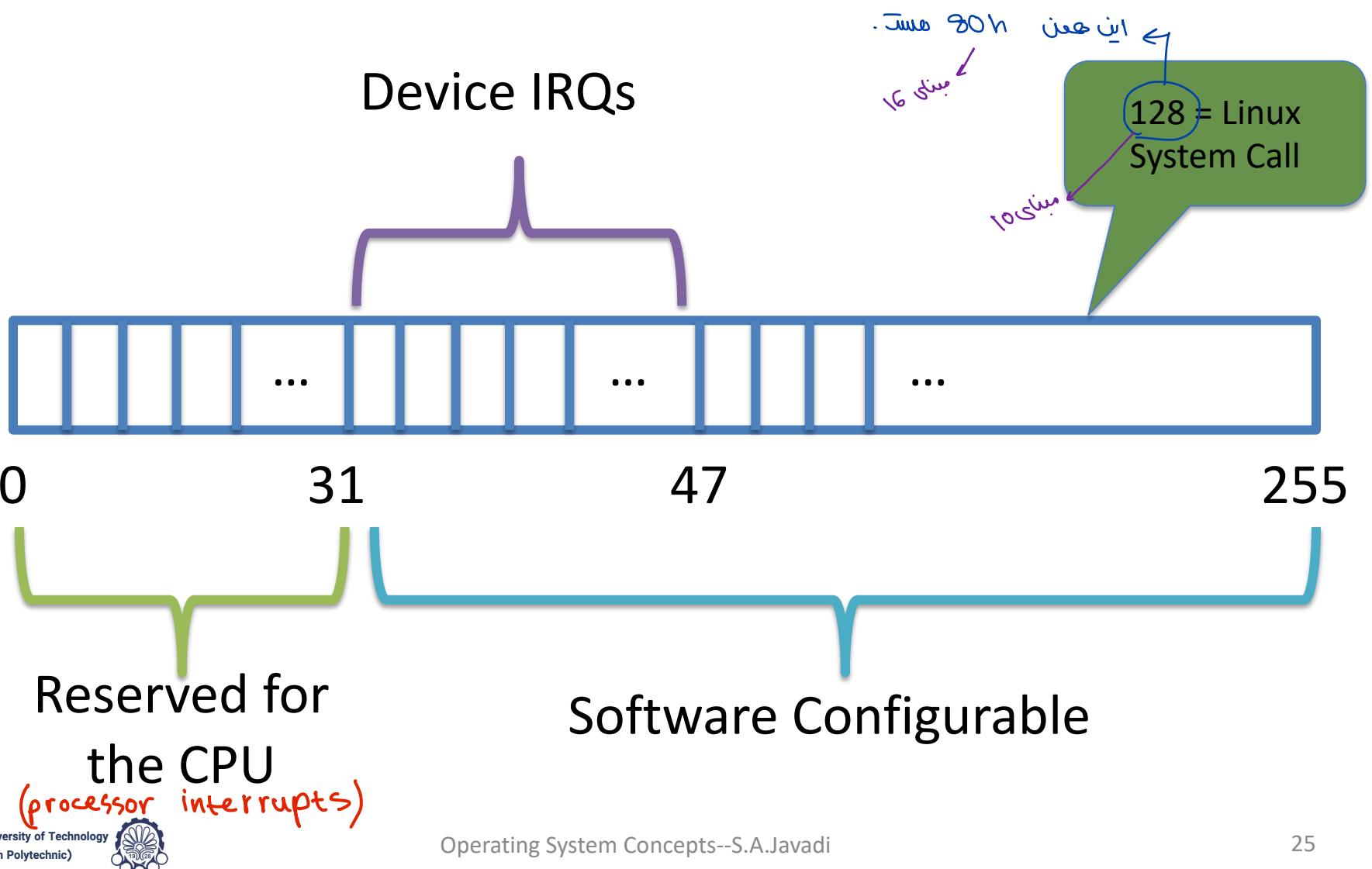
x86 interrupt table

- ... , flash controller, disk controller $\xrightarrow{\text{IRQ}}$

Device IRQs (Interrupt ReQuests)



x86 interrupt table



x86 interrupt overview

- Each type of interrupt is assigned an index from 0—255.
- 0—31 are for processor interrupts; generally fixed by Intel
 - E.g., 14 is always for page faults
- 32—255 are software configured
 - 0x80 issues system call in Linux (more on this later)
80h

Software interrupts

برای مدارزن وقفه نرم افزاری

- The `int <num>` instruction allows software to raise an interrupt
→ توسعه دهنگان Linux آن را 80h و ۰x80 به عنوان یک system call برای استفاده کنند.
- There are a lot of spare indices
→ ممکن است ۲۰۰ حرفه را جدید اضافه کردن و تابلیت جدیدی .w u Kernel
- You could have multiple system call tables for different purposes or types of processes!
→ Windows does: one for the kernel and one for win32k



What happens (generally):

در هنگام دفعه، پی آنفاص رخ می‌دهد؟

- Control jumps to the kernel

* توی جدول منگار برای `int` هفتاد خلاصه بالاً تجارتی. سعی ماهر ریم کده انجینیرین.

- At a prescribed address (the interrupt handler)

* وقتی `NN` exception حدث شود، کورتیس سست Kernel سه من باید اول از همه از خارج و `eg` طای CPU نیز `Jump` کنم
تا وقتی `NN` هنوز نشود، برگردیم به ادامه اجرای این محتواهای قابل آن توی محاسبات را شنید و داشتن بالسلام بخواهیم
از آنای `copy` مانند توی `reg` طای CPU، موتور بپلست `load` می‌شنیم.

- The register state of the program is dumped on the kernel's stack

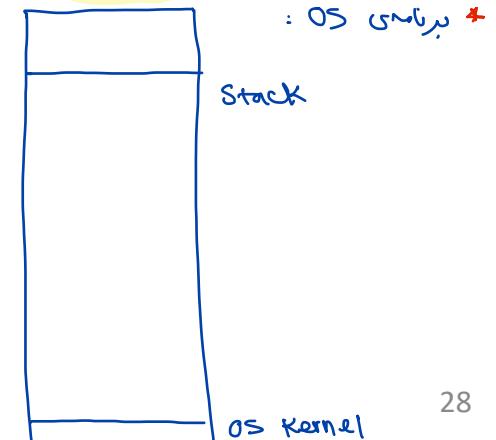
* وقتی داین تغییری از لذتمنی سه مسلسل از `reg` های توی CPU است.
`P1 reg P2 N P1 reg P2 N P1 reg P2` با `N` باید `disk` \rightarrow `ram` باشد.
* وقتی `NN` می‌شود و تا برنامه داریم `P1` \rightarrow `P2` را باید از لذتمنی `reg` \rightarrow `load` برخواهد. این میکنیم `Context Switch` می‌کنیم.

- Kernel code runs and handles the interrupt

* جون `eg` طایی جائز نماید که سیستم برگردی به برنامه کنیم
* سین `eg` طایی داریم `load` کدن و `lret` را در دست برنامه.

- When handler completes, resume program (see `iret` instr.)

* هر برنامه‌ی توی مقتای خودستی پسند تعریف می‌کند. `N` حکمیت کالی تغییره.
برنامه OS این `eg` هاردوئی Kernel's stack ذخیره می‌کند.



* در حلول برگرداننده کل دفعه نایاب و تغییراتی دخواه، این رخداد OS خواهد می‌کند.

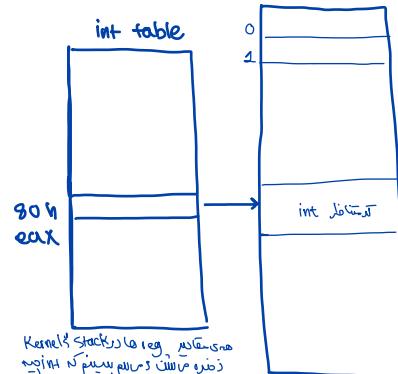
آن این آنفاص بیرون، `NT!CrashHandler`



System call “interrupt”

* وقتی تری برناه سیستم هرچیزی نخارج از برنامه ماهست بیاره بناره نو رو دهندگانه کنیم.

- Originally, system calls issued using int instruction
- send out , ارسال
- Dispatch routine was just an interrupt handler
- Like interrupts, system calls are arranged in a table
 - See arch/x86/kernel/syscall_table*.S in Linux source
- Program selects the one it wants by placing index in eax register
 - Arguments go in the other registers by calling convention
 - Return value goes in eax



* موقعیت داخل func نیز main را میداریم زیرا یک این jump کردن تا func نیز reg N هارو جایی ذخیره کنیم . این jump کردن تا func نیز دخواستن در سیستم می باشد این system call را میداریم ،
این کار برای اجرا نمایند ، باید این اجرا بوسیله reg داده شود .
برنامه از اینجا آغاز می شود . اینجا داده های سیستم را ذخیره کنیم .

