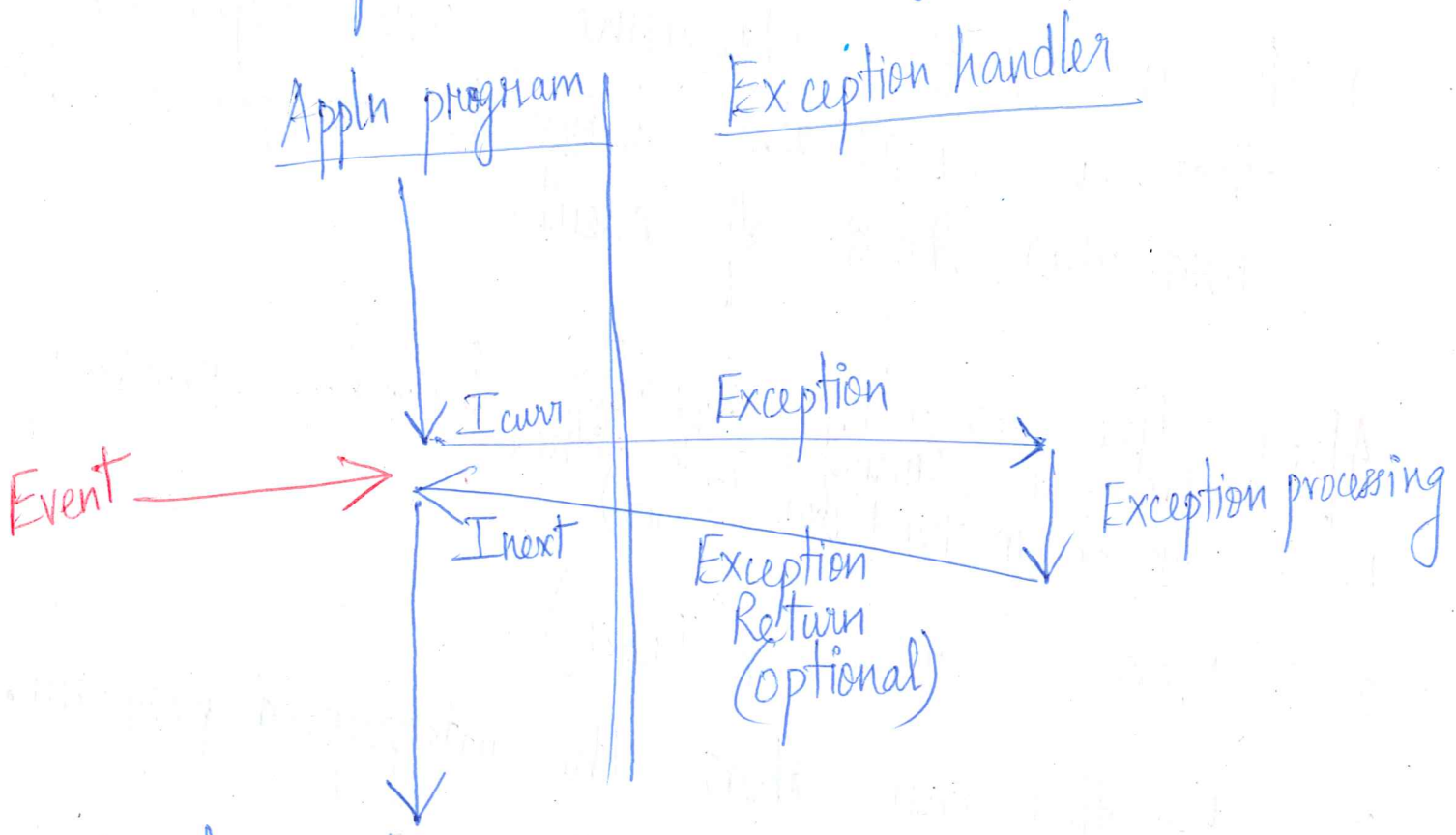
Exceptional Control Flow

Exceptions

Exception - an abrupt change in the control flow in response to some change in the processor's state.



Examples for events

1. VM page fault
2. Arithmetic overflow
3. Divide by zero.
4. System timer goes off.
5. I/O request completes

} unrelated to the exec. of the curr instruction.

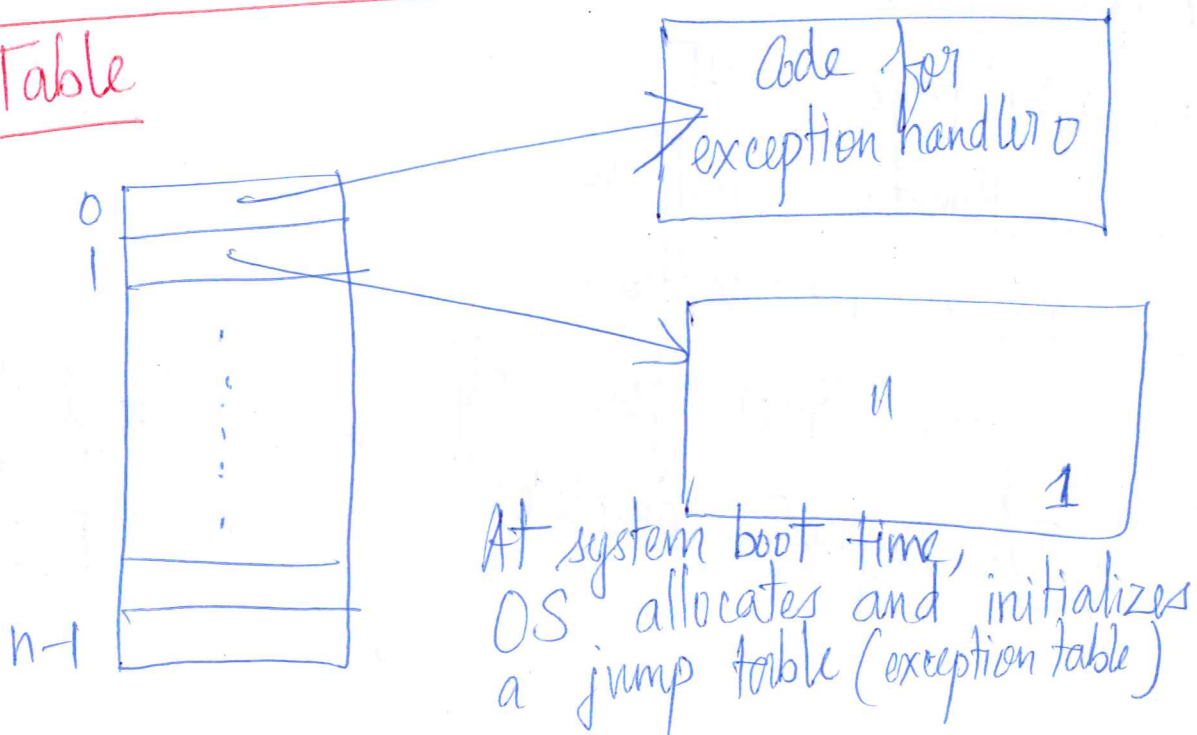
When the processor detects that the event has occurred:

1) It makes an indirect procedure call (the exception).
2) through a jump table called an exception table.
3) to an OS subroutine (the exception handler) that is specifically designed to process this particular kind of event.

After the exception handler finishes processing, control can transfer to either.

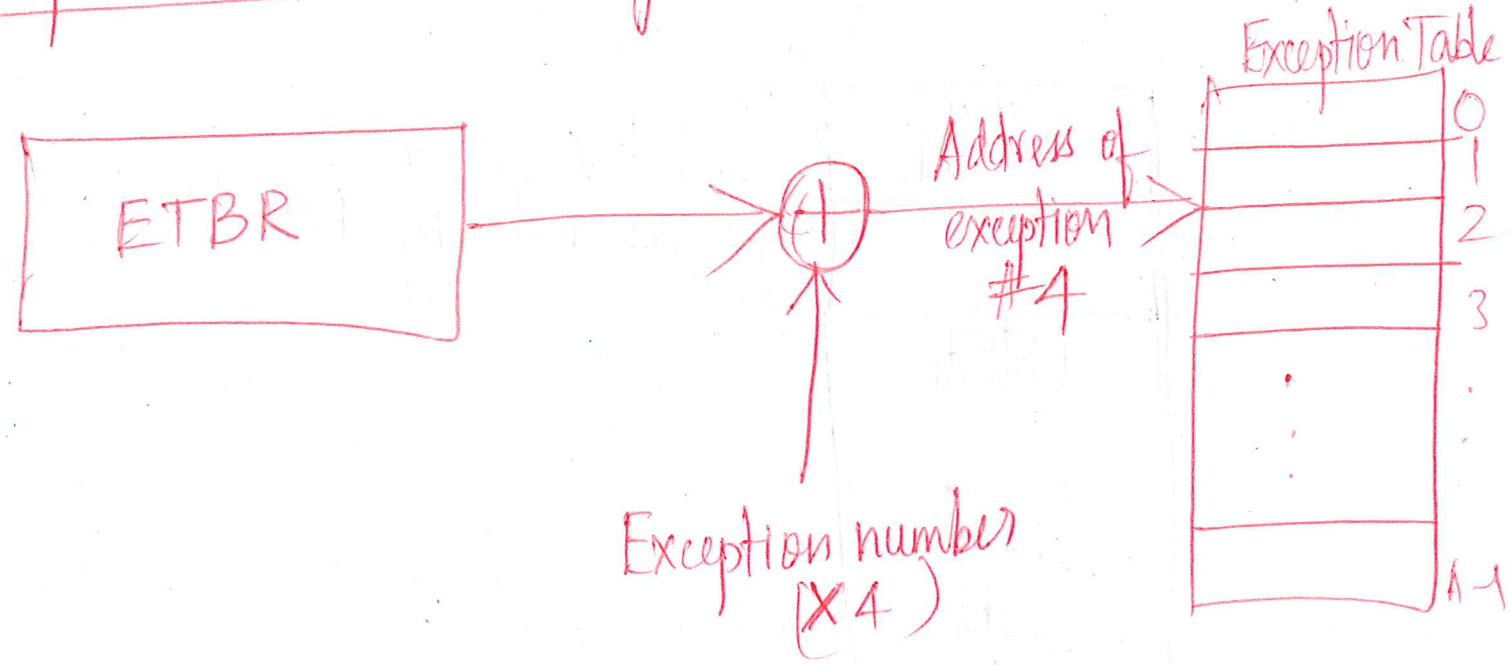
1. Current instruction (I_{curr})
2. Next " (I_{next})
3. Handler can abort the interrupted program.

Exception Table

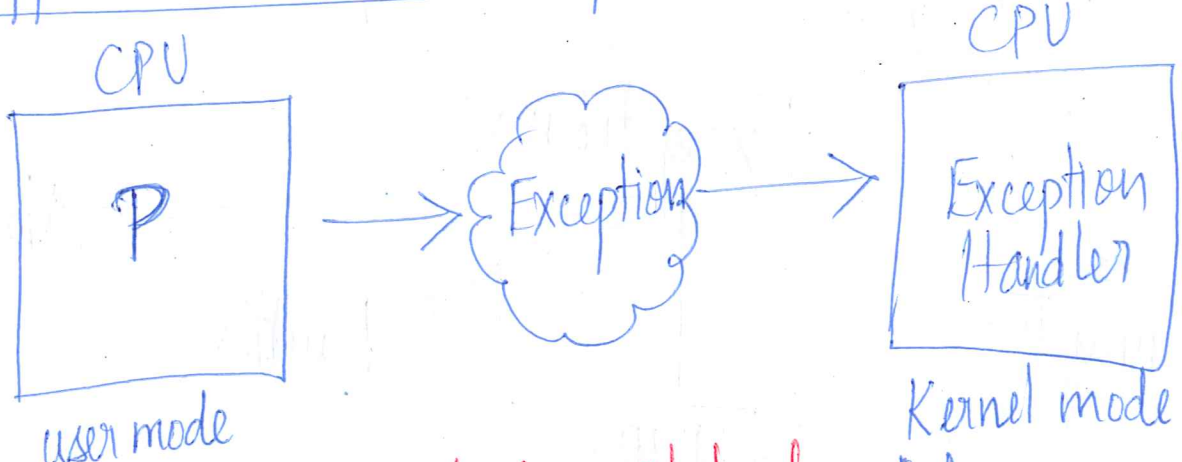


At system boot time, OS allocates and initializes a jump table (exception table)

Exception Table Base Register



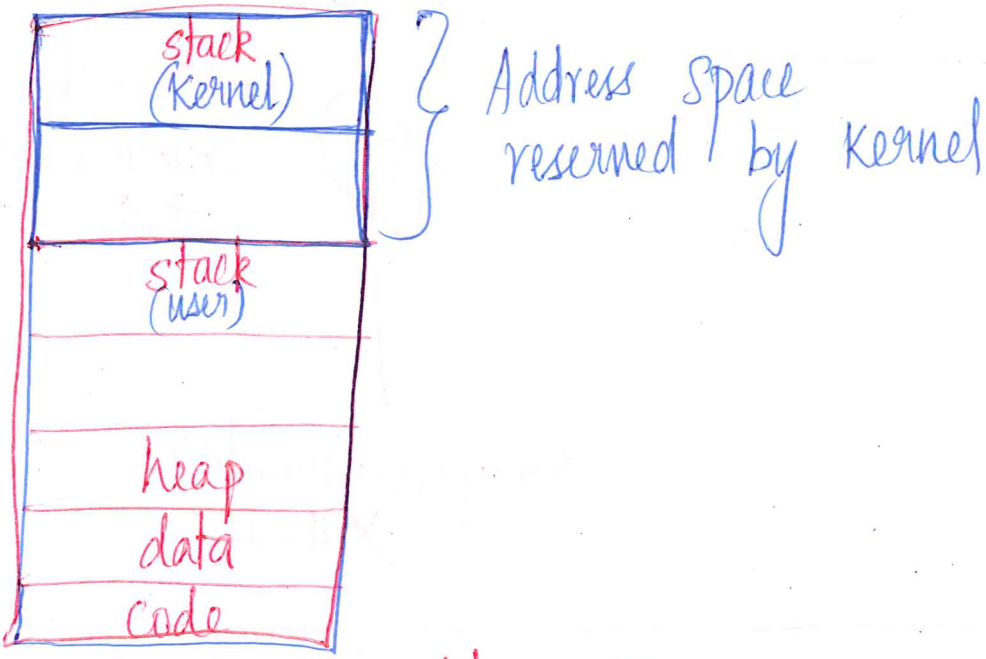
What happens when an exception occurs?



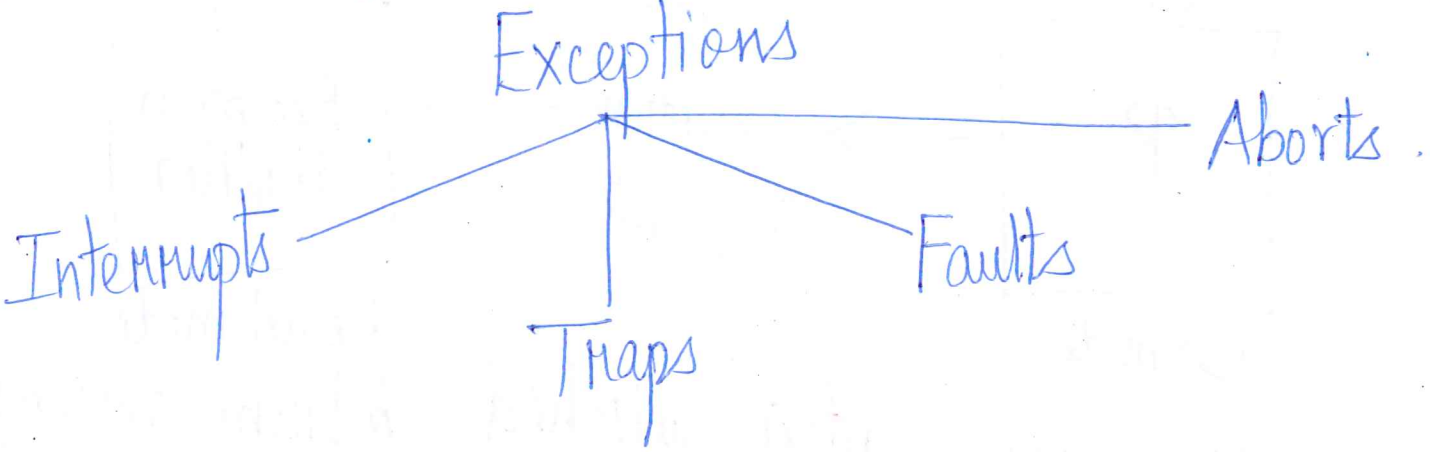
The CPU has context switched from running the user program P to running the exception handler.

X CONTEXT SWITCH

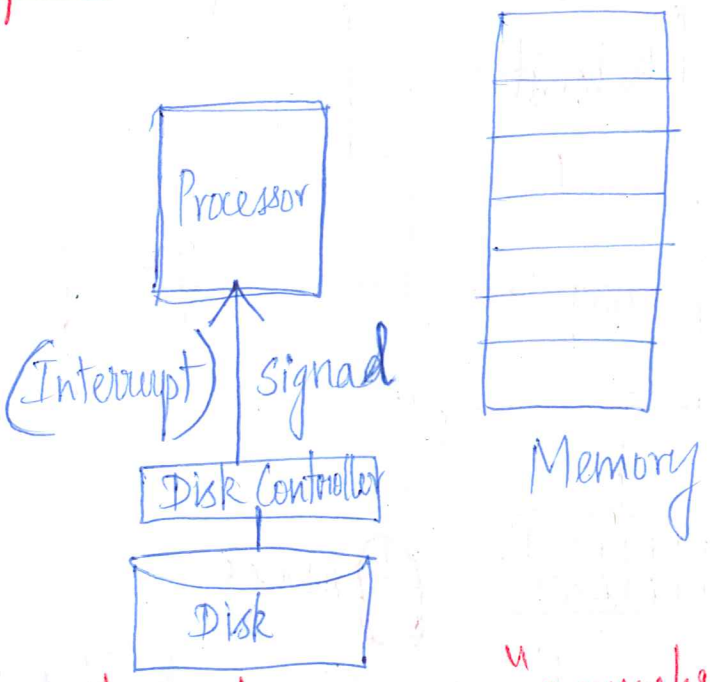
User Stack vs Kernel Stack



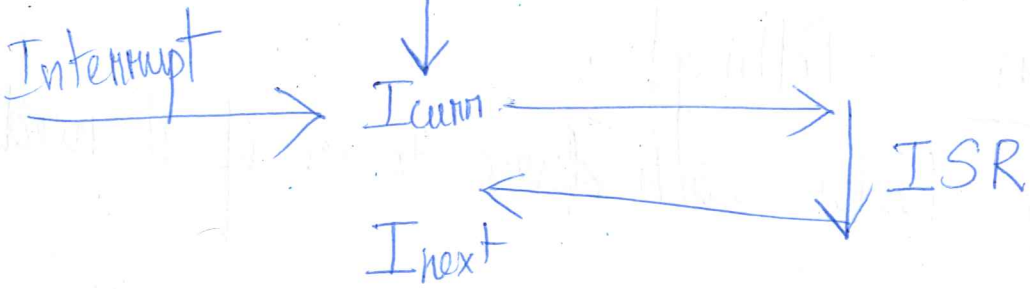
A single process' address space.

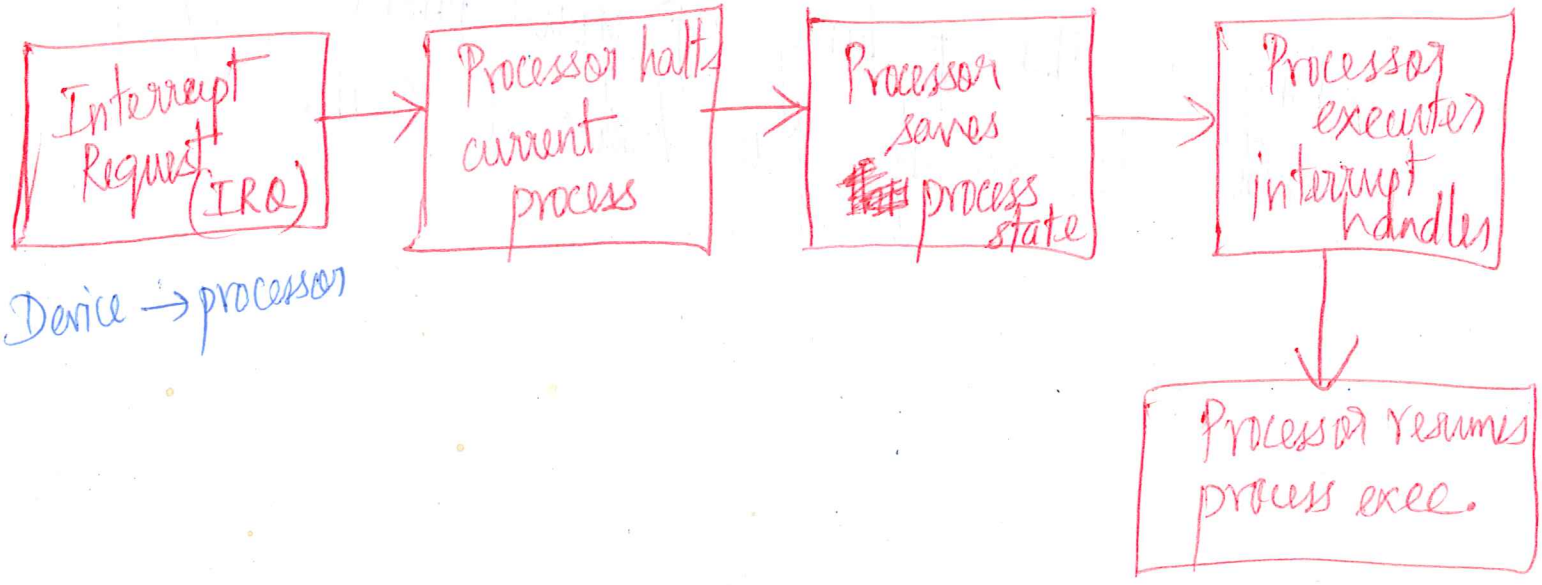


Interrupts



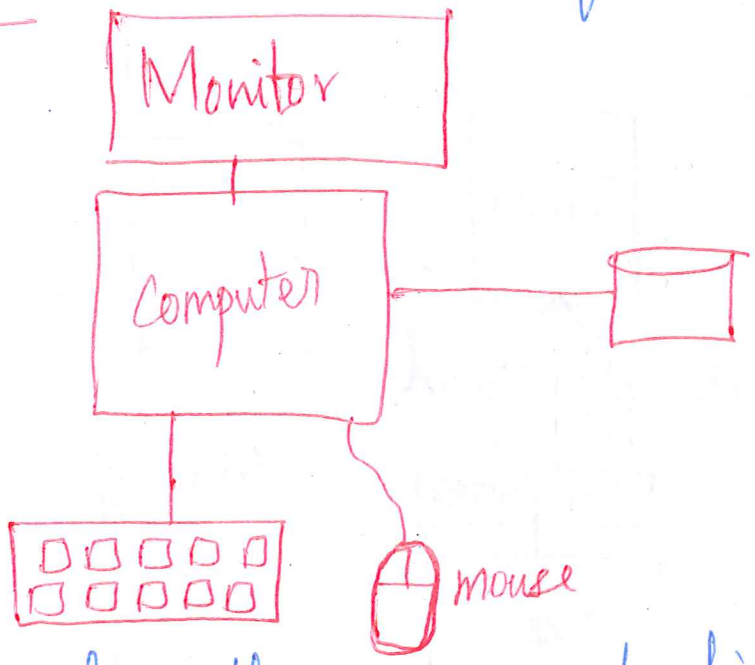
Hardware interrupts are "asynchronous" w.r.t. the current process.





Why interrupts?

Ref: U of T slides on interrupts



* Need a way for the CPU to find out when devices need attention.

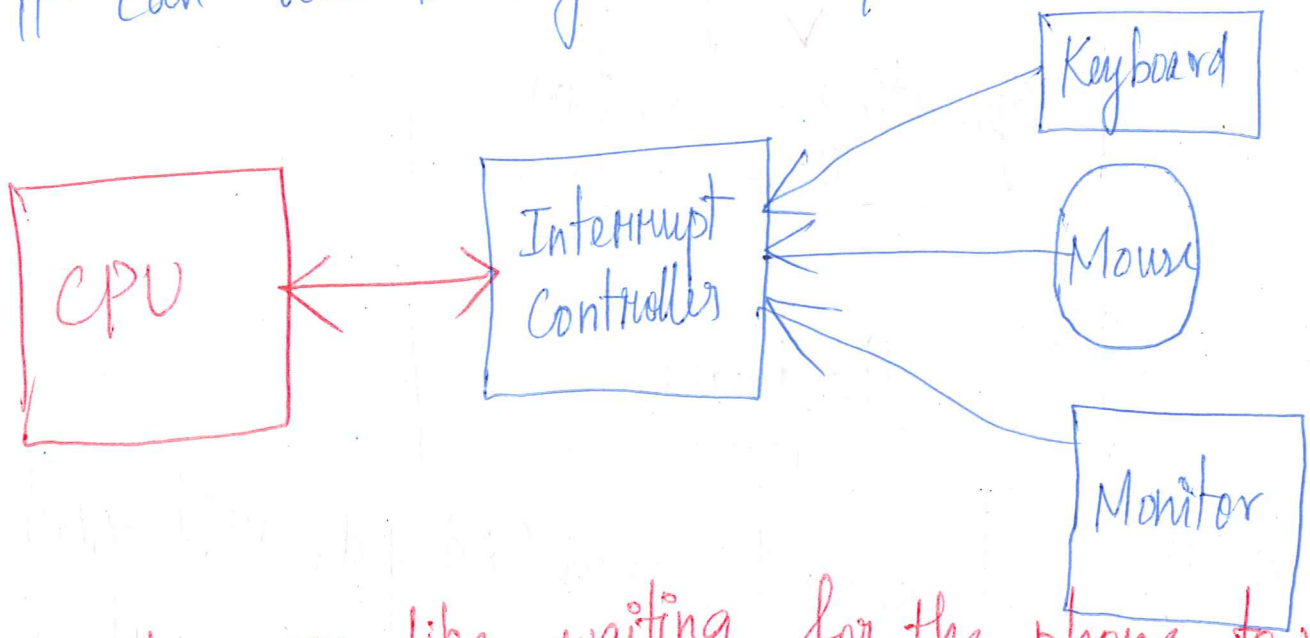
Possible Solution - Polling!

CPU periodically checks each device to see if it needs service.

"Polling is like picking up your phone every few seconds to see if you have a call."

Alternative: Interrupts

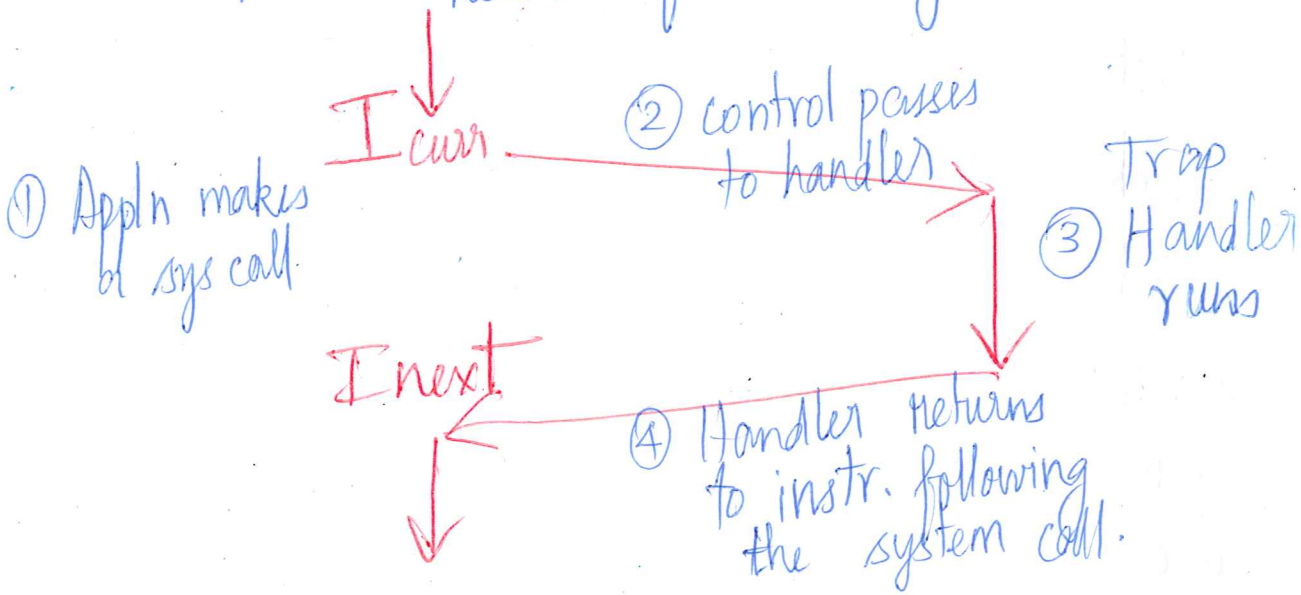
Each device is given a wire (interrupt line) that it can use to signal the processor.



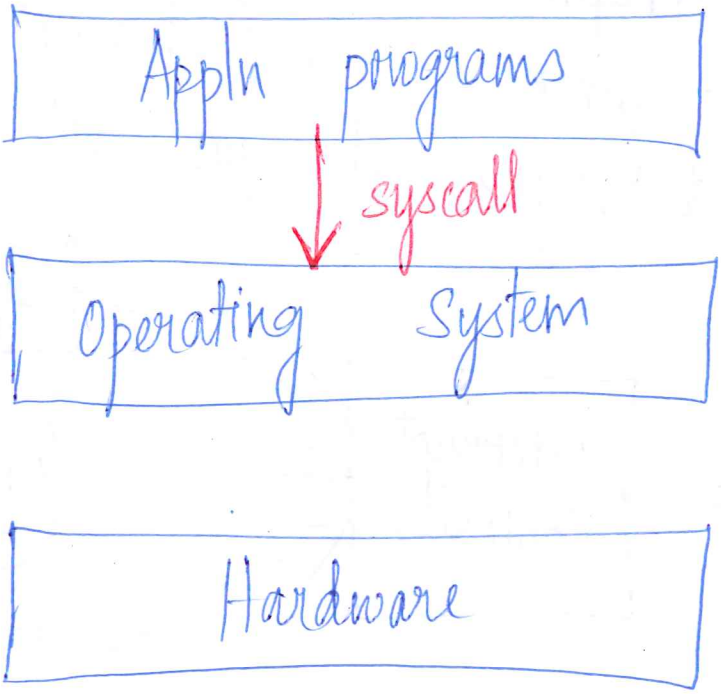
"Interrupts are like waiting for the phone to ring!"

Traps and System Calls

Traps - intentional exceptions that occur as a result of executing an instruction.



System Call



1. open
2. read
3. write
4. close

```
ssize_t read(int fd, void *buf,
             size_t count);

"      write(      "      );
```

#	Name
1	exit
2	fork
3	read
4	write
5	open
6	close

Linux / IA 32 System Calls

int n

int → interrupt

n → exception number.

n = 0x80 (128_{10}) for system calls.

```
int main()
{
    write(1, "hello, world\n", 13);
    exit(0);
}
```

Syscall # → stored in %eax.

arguments → stored only in registers and not on stack.

%ebx, %ecx, %edx, %esi, %edi, %ebp

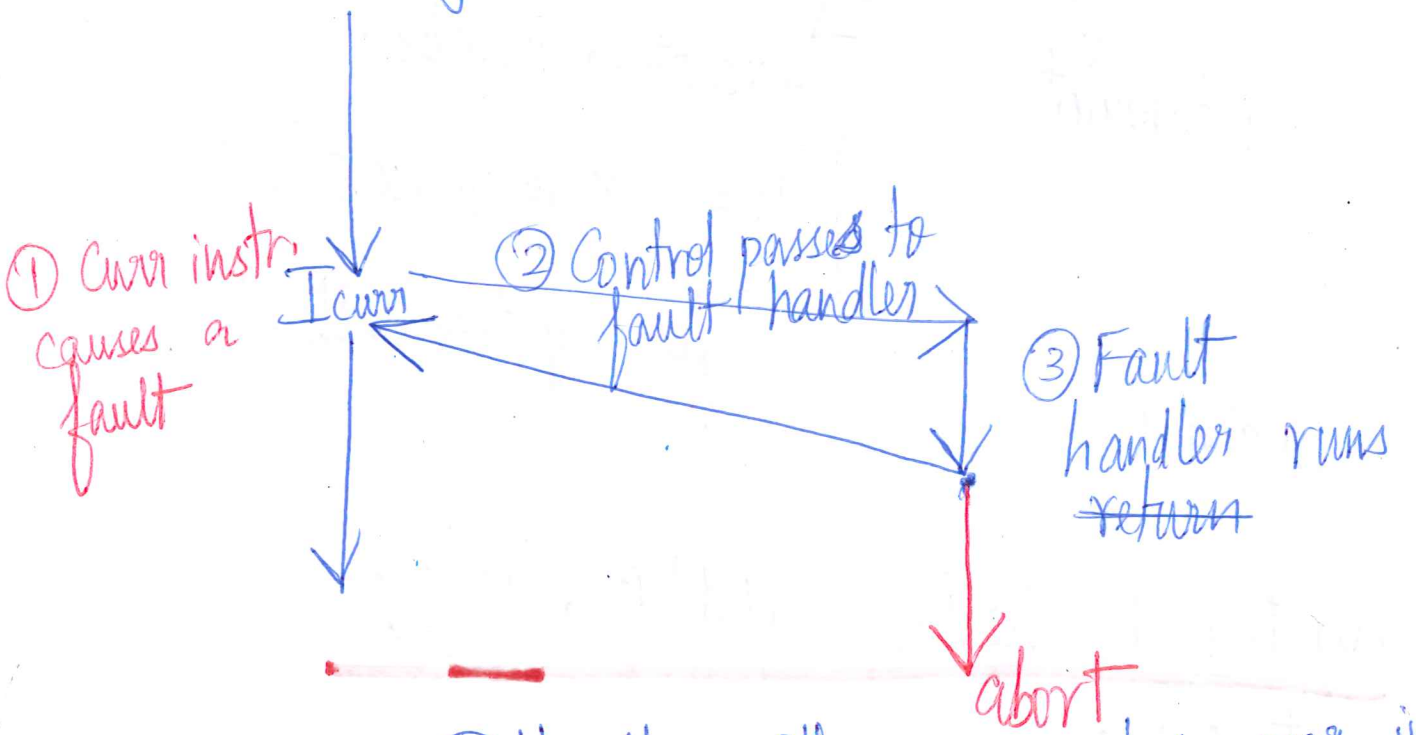
%ebx → arg1

%ebp → arg6.

%esp will not be used.

Faults

Potentially recoverable error.



④ Handler either reexecutes curr instr. or aborts.

eg. page fault exception, segmentation fault.

Exception #	Description	Exception class
0	Divide error	Fault
13	General protection fault	"
14	Page fault	"
18	Machine check	Abort
128 (0x80)	System Call	Trap.
other # (max 255)	OS-defined exceptions	Interrupt or Trap.

```
.section .data
string:
    .ascii "hello, world\n"
string_end:
    .equ len, string_end - string

.section .text
.globl main
main:
    # First, call write(1, "hello, world\n", 13)
    movl $4, %eax          # System call number 4
    movl $1, %ebx          # stdout has descriptor 1
    movl $string, %ecx     # Hello world string
    movl $len, %edx        # String length
    int $0x80              # System call code

    # Next, call exit(0)
    movl $1, %eax          # System call number 0
    movl $0, %ebx          # Argument is 0
    int $0x80              # System call code
```

data segment

str

str end

assembler directives.

code segment.

int → interrupt.