I/O & Interrupts

COMP 310 Operating Systems

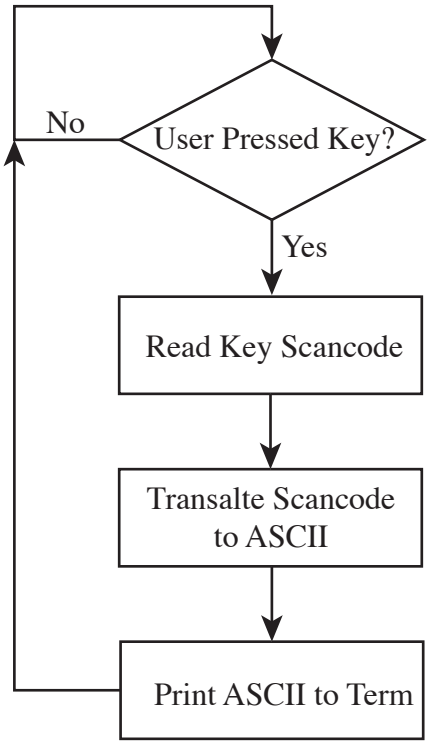
January 28, 2020

1 Introduction

Problem: How to communicate with external devices?
From the computer's perspective, we only have a CPU and memory.

- Special addresses in memory reserved to interact with hardware: sending commands, reading/writing data, etc.
- Interrupts from the external devices tell the computer that it needs attention.

1.1 Basic Example: Polling



I/O Port	Access Type	Function
`0x60`	R/W	Data Port
`0x64`	R	Status Register
`0x64`	W	Command Register

1.1.1 PS/2 Status Register

7	6	5	4	3	2	1	0
PE	TOE	UNK	UNK	CD	SF	IS	OS

PE	**Parity Error**	0 indicates no error, 1 indicates error
TOE	**Timeout Error**	0 indicates no error, 1 indicates error
UNK	**Chipset Specific**	
UNK	**Chipset Specific**	
CD	**Command/Data**	0 indicates data written to input buffer is data for PS/2 device, 1 indicates data written to input buffer is data fro PS/2 Controller
SF	**System Flag**	Set by BIOS if system passes self test (POST)
IS	**Input Buffer Status**	0 means empty, 1 means full
OS	**Output Buffer Status**	0 means empty, 1 means full

1.1.2 Reading and Writing to IO Ports on x86

On x86, we have two special instructions to communicate with IO ports: `in` and `out`.

To determine if the user pressed a key, we want to read the PS/2 status register and check the LSB. If the LSB is 1, then that means that means that the PS/2 controller's output buffer contains a scancode. To get the scancode, read from IO port `0x60`.

```
.lp:
    in ax,0x64          ; Get keyboard status reg
    and ax,1
    je .lp              ; If LSB in status is clear, no key
    in al,0x60          ; Get scancode
    mov ah,0
    push ax
    call print_hex_16   ; Print the scancode
    add sp,2
    jmp .lp             ; Check for another keypress
```

Question: is this a good approach?

What happens when you have more than one I/O device? What about if you have some other tasks running that don't involve I/O? Will this approach affect responsiveness?

2 Interrupts

Interrupts are functions that are **called by the hardware** to handle IO. The interrupt handling process is basically the same for all processors, but the details of the data structures are different. Here's how the process works in general:

- There is an array of pointers called a vector table in memory somewhere. Location is defined by the CPU manufacturer.
- Each entry in the array points to a function that handles a particular IO event.
- When an IO event occurs:
  1. We stop executing the user code
  2. Push an interrupt stack frame onto the stack
  3. Jump to the address in the vector table and run the interrupt handler.
  4. Pop the interrupt stack frame and return to user code.

Interrupt Stack Frame

SP+4	`USR Flags`
SP+2	`USR CS`
SP+0	`USR IP`

2.0.1 Installing an Interrupt Vector

```
boot:

    ; Print greeting string
    push greeting
    call putStr
    pop ax

    lea ax,[keyboard_hook]  ; Get addr of keyboard handler in AX
    mov [0x24],ax           ; Install kb handler
    mov word [0x26],0       ; Install kb handler CS
.lp
    hlt

    jmp .lp                   ; Loop forever

keyboard_hook:
    pusha                ; Save all user registers on the stack

    mov dx,[num_key_presses]
    inc dx
    mov [num_key_presses],dx

    in al,0x60           ; Get keyboard scancode in AL

    push key_pressed_msg ; Print message about key being pressed to the screen
    call putStr
    add sp,2

    mov al,0x20          ; We need to write 0x20 to IO port 0x20 in order to
    out 0x20,al          ; acknowledge the interrupt

    popa                 ; Restore user registers
    iret                 ; Return
```

Interrupt #	Address	Contents
119-255	0x01DE	Software Interrupts
118	0x01DA	Hard Disk ISR CS
118	0x01D8	Hard Disk ISR IP
117	0x01D6	Math Coprocessor ISR CS
117	0x01D4	Math Coprocessor ISR IP
116	0x01D2	PS/2 Mouse ISR CS
116	0x01D0	PS/2 Mouse ISR IP
115	0x01CE	Reserved
115	0x01CC	Reserved
114	0x01CA	Reserved
114	0x01C8	Reserved
113	0x01C6	Redirected ISR CS
113	0x01C4	Redirected ISR IP
112	0x01C2	Real Time Clock ISR CS
112	0x01C0	Real Time Clock ISR IP
16-111	0x0040	Software Interrupts
15	0x003E	Parallel Port ISR CS
15	0x003C	Parallel Port ISR IP
14	0x003A	Floppy Disk ISR CS
14	0x0038	Floppy Disk ISR IP
13	0x0036	Reserved/Sound Card ISR CS
13	0x0034	Reserved/Sound Card ISR IP
12	0x0032	Serial Port 1 & 3 ISR CS
12	0x0030	Serial Port 1 & 3 ISR IP
11	0x002E	Serial Port 2 & 4 ISR CS
11	0x002C	Serial Port 2 & 4 ISR IP
10	0x002A	Redirected ISR CS
10	0x0028	Redirected ISR IP
9	0x0026	Keyboard ISR CS
9	0x0024	Keyboard ISR IP
8	0x0022	System Timer ISR CS
8	0x0020	System Timer ISR IP
7	0x001E	ISR CS
7	0x001C	ISR IP
6	0x001A	ISR CS
6	0x0018	ISR IP
5	0x0016	ISR CS
5	0x0014	ISR IP
4	0x0012	ISR CS
4	0x0010	ISR IP
3	0x000E	ISR CS
3	0x000C	ISR IP
2	0x000A	ISR CS
2	0x0008	ISR IP
1	0x0006	ISR CS
1	0x0004	ISR IP
0	0x0002	ISR CS
0	0x0000	ISR IP