What is an OS?

Examples: OSX
           Windows
           Solaris
           Linux
           QNX
           PalmOS
           BSD

Common things that an OS do:  - initialize HW
                              - File System
                              - HW Interface

For a web server (for eg.) it does:  - resource management
                                    - etc.

HW ——— [OS] ——— Program
     (set of executable instructions)

Consider:

* 16-bit microprocessor
* 16-bit addresses and instructions
* R0 - R7   (registers)
* SP        (Stack pointer)
* IP        (Instruction Pointer)
Instruction:

- MOV val, val
- ADD/SUB val, val
- JMP (condition) val
- CMP val, val
- CALL addr

\[ \text{val} = R_x \quad (\text{Register}) \]
\[ = * (\text{addr}) \]
\[ = * (R_x) \]
\[ = * (R_x + \) \]

Example:

- MOV 5, R6
- MOV R6, *(R5++)

Address Space

RDY = 1 if key is pressed
it is reset to 0 once the value of key is read from KEY.

F801 \[ \text{RDY} \]
F800 \[ \text{KEY} \]

24x80 bytes Frame Buffer

text displayed on screen (for eg.)
CSG712 - INTENSIVE COMP SYSTEM

PROGRAM 1:  
```
    str = 'H', 'e', 'l', 'l', 'o'
    count = 5

    mov str, R1
    mov F000, R2
    mov count, R3

    Loop:   mov *(R1++), R4       Loop2:
             cmp *(F802), 0
             jmp (EQ), Loop
             sub 1, R3
             jmp (NZ), Loop

    Done:    Halt
```

This program prints "Hello" on the Frame Buffer.

PROGRAM 2:  
```
    str = 'H', 'e', 'l', 'l', 'o'
    count = 5

    mov str, R1
    mov count, R2

    Loop:    call getchar
             mov *(R1++), R3
             push R3
             call putchar
             sub 1, R2
             jmp (NZ), Loop
```
Program 2 does the same thing except that we have added subroutines of `GETKEY` and `PUTCHAR`.

These subroutines are part of the OS. The program knows the address of these subroutines in the address space of the operating system. But notice that we cannot upgrade the operating system for this reason or we cannot relocate it.

This way we have

- Common HW interface functions
- Separating programs and OS in memory
- Allows loading programs.
Since upgrading OS is a problem because the programs have hardcoded address to subroutines, we add a table of addresses / vector table / system call table in the beginning of memory.

![Diagram of function call with addresses]

```c
#define GETKEY = 0
mov *(GETKEY), RO
CALL RO
```

The table of addresses contains the addresses of the subroutines (hardware interface functions) in the OS memory.

Adding this table of addresses, we can upgrade/relocate the OS without any need to change the program source code. We can also use the same program later with a different version of the OS (given backward compatibility).
CSG712 - INTENSIVE COMP. SYSTEMS.

In 80x86 INT <n>

Address: N*4

This is similar to hardware interrupts in 8086.

We can, therefore, independently change OS and program.

Adding a disk drive:

We have added some more registers. We may have an instruction READSECTOR n
where n is the sector number.

We set

SECTORS = n
read/write = read
GO = 1

we wait until RDY == 1 , then read the data block.
In earlier IBM PCs, we get the key from keyboard (call-5) using the interrupt INT 16, the key read is stored in one of the registers. INT 10 is used to display characters (from one of the registers) on the screen.

We now have a single user OS.

- OS and programs are independent
- Multiprogramming is missing
- Resource management is missing
- Security - We can store OS in the readonly memory but the data has to be stored in the other memory, therefore it is not secure.

Part of the program:

- Code (instructions)
- Fixed size data (global/static variables)
- Dynamically allocated memory (heap)
- Stack

```
+---+  +---+  +---+
<table>
<thead>
<tr>
<th>OS</th>
<th></th>
<th>Stack</th>
<th></th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Code &amp;</td>
<td></td>
<td>Fixed data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vector table</td>
</tr>
</tbody>
</table>
```