Syllabus (Topics that would be covered during the course)

- Hardware models
- Programs
- Program Interface (OS API)
- Processes
- Threads and synchronization
- Memory management
- Virtual Memory
- I/O device interfaces
- Virtual machine (H/W virtualization)
- Disks and storage
- File systems
- Buffering, caching, replacement
- SMP & NUMA (Non uniform memory access)
- Program loading
- Security
Various examples of Operating Systems

Windows Vista, XP
Linux
Solaaris
OS X
Symbian
Windows Mobile
Palm OS
M6 DOS
Freedas

What is an operating system?
- It is evolving and a moving target
- Helps in managing resources
- It provides a platform for running applications

Thus an OS is a set of one or more programs which provides a set of services that interface applications to computer hardware and which allocates and manages resources shared among multiple processes.

hardware $\longleftrightarrow$ [OS] $\longleftrightarrow$ program
Program 1: Prints Hello on the Screen

16 bits

R1

FF 00

R2

= 2000 bytes

R3

EOF

R4

× 80 columns

SP

25 rows

PC

0000

Buf = F000

Str: 'H' 'e' 'l' 'l' 'o'

mov Str, R1
mov buf, R2
mov 5, R3

Mov addr
mov
F000

'Hi' 'e'
'1' 'l'
'0'
loop:
    mov *(R1++), R4
    mov R4, *(R2++)
    sub 1, R3
    jmp (not zero) loop
done:
    jmp done

Execution:
Buf = F000
Str = 'H' 'e' 'l' 'l' 'o'

<table>
<thead>
<tr>
<th>Before start of execution</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>'H' 'e' 'l' 'l' 'o'</td>
<td>F000</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>'e' 'l' 'l' 'o'</td>
<td>H 3</td>
<td>4 4</td>
<td>H 4</td>
</tr>
<tr>
<td>loop</td>
<td>'l' 'l' 'o'</td>
<td>e</td>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>loop</td>
<td>'l' 'o'</td>
<td>l</td>
<td>2</td>
<td>l</td>
</tr>
<tr>
<td>loop</td>
<td>'o'</td>
<td>l</td>
<td>1</td>
<td>l</td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Program 2: Prints what the user has typed (up to 5 characters) on the keyboard (i.e., if keyboard is ready and key is pressed).

```
mov str, R1
mov buf, R2
mov 5, R3
```
loop:
  mov RDY, R4
  cmp R4, 0
  jmp (z), loop

mov *(R1++), R4
mov R4, *(R2++)
sub 1, R3
jmp (nz), loop

wait for any key
print one character
if we're not done, loop

done: jmp done

What does this OS do?
- It manages access to the H/w

What does it not do?
- It doesn't allow loading separate programs.
- It doesn't provide binary compatibility.
- It doesn't allow upgrading OS.
- It doesn't provide security.

This material goes on the next page, after Program 3. Note that the "OS" there is just the GETKEY and PUTKEY functions.
Program3: It is same as Program2 with modifications to loop. It uses GETKEY and PUTCHAR subroutines which can be part of Program Space.

16 bits

R1 FFFF

R2 Food

R3

R4

SP GETKEY

PC PUTCHAR

0000

mov str, R1
mov buf, R2
mov 5, R3
loop:
    call GETKEY
    mov * (R1 +), R4
    push R4
    call PUTCHAR

done:  jmp done

Program: separate OS from program

- The advantage of this is that it allows multiple possible programs.
- The disadvantage is that programs are tied to 0.6 internal addresses. (So if
Any OS upgrades are done on the hardcoded addresses are changed then the programs won't work, since the programs depend on the OS addresses.

The solution for this would be to use a OS jump table:

**Programs**: Use OS jump table.

Thus the jump table makes the program independent of OS internal addresses.
Program 6: OS + Disk read/write

0000

F801 Rdy
E800 Val

Keyboard

E206 Go
E204 Rdy
E202 Is_Write
E200 Blk_Num

Buffer + Disk drive

E000 - E1FF: Sige byte buffer

R/W
read block 100

BLK_NUM = 100
IS_WRITE = false
GO = true

while (!RDY)
    wait .... ;

now block 100 is in buf [0.... 511]

Summary:

Program 1 & 2 → Direct hardware access
   Prog 1: Write Only
   Prog 2: Wait for key before writing

Program 3 → Link OS (GETKEY, PUTCHAR) with the program

Program 4 → OS is separate from program
- This allows multiple possible programs
- But programs are tied to OS internal addresses.
Program 5 —> Use an OS jumtable

Program 6 —> OS + disk read/write