CHAPTER 1

OS FEATURES

Dr. Soha S. Zaghloul

LAYOUT

OS Features

Multiprogramming

Multitasking

Multiple Operational Modes

Timers

OS FEATURES

OS Features

A modern OS is provided with the following features:

- Multiprogramming
- Multitasking (Time sharing)
- Multiple Operational Modes
- Timers

MULTIPROGRAMMING (I) – INTRODUCTION

A single program cannot make full use of the computer resources: these include the CPU and the I/O devices,

This results in a low percentage of resources utilization, and therefore a low performance.

Multiprogramming increases CPU utilization by organizing jobs (programs) so that the CPU almost always has one to execute.

OS FEATURES

MULTIPROGRAMMING (2) – TECHNIQUE

The OS applies multiprogramming as follows:

1. Initially, all jobs ready for execution are kept on the disk in the job pool.

   The job pool includes all processes (programs) that are ready for execution, and are awaiting for memory allocation.

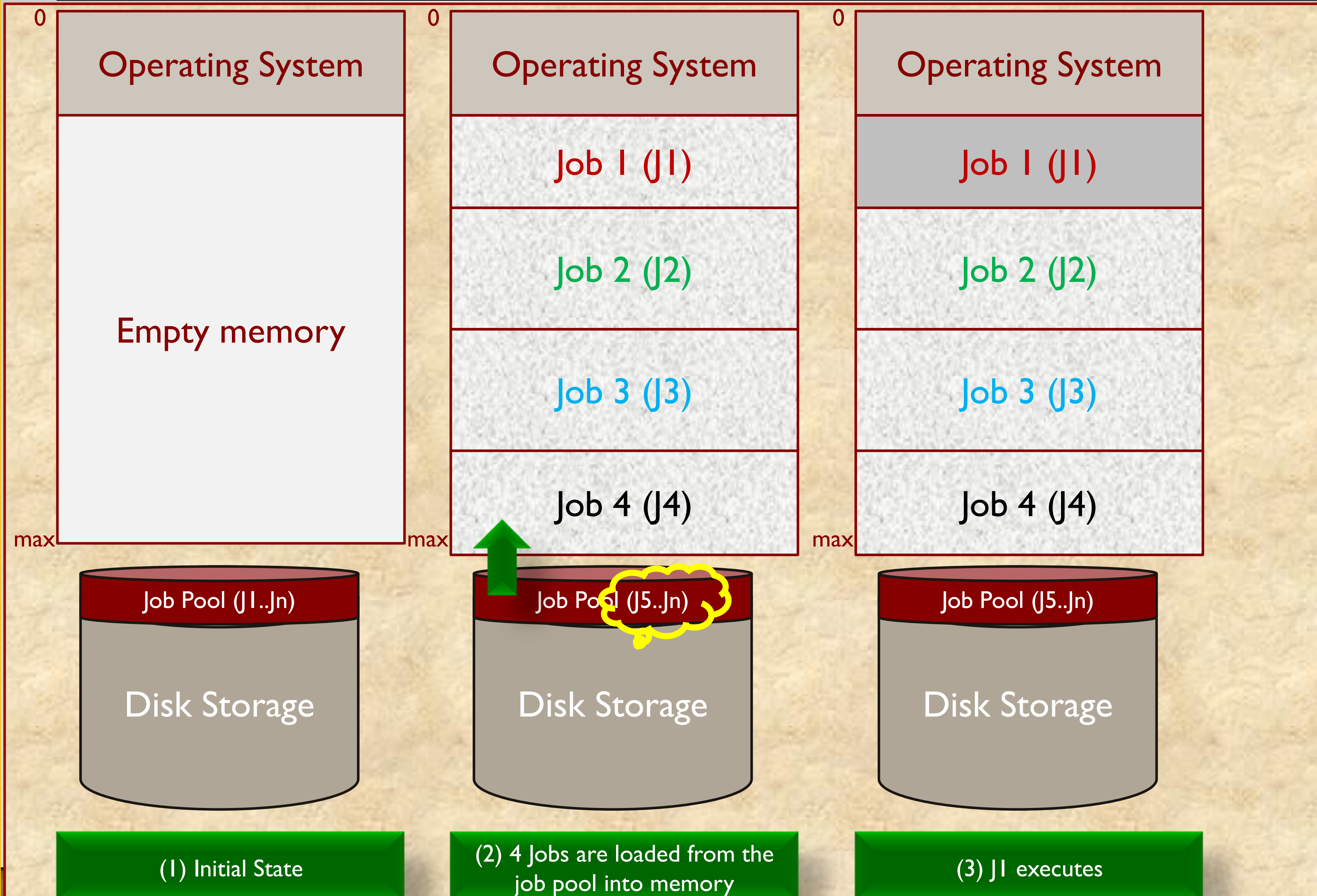
2. The OS selects several jobs according to a policy, and allocates them into memory.
3. The OS picks one of the jobs ***(j1)*** in the memory and starts its execution. Selection is made according to a specific preset policy.
4. While running, the job ***(j1)*** may have to wait for some task, such as an I/O operation, to complete

   In a non-multiprogrammed environment (ie. where the OS is not designed to handle more than one program at a time), the CPU would remain idle.

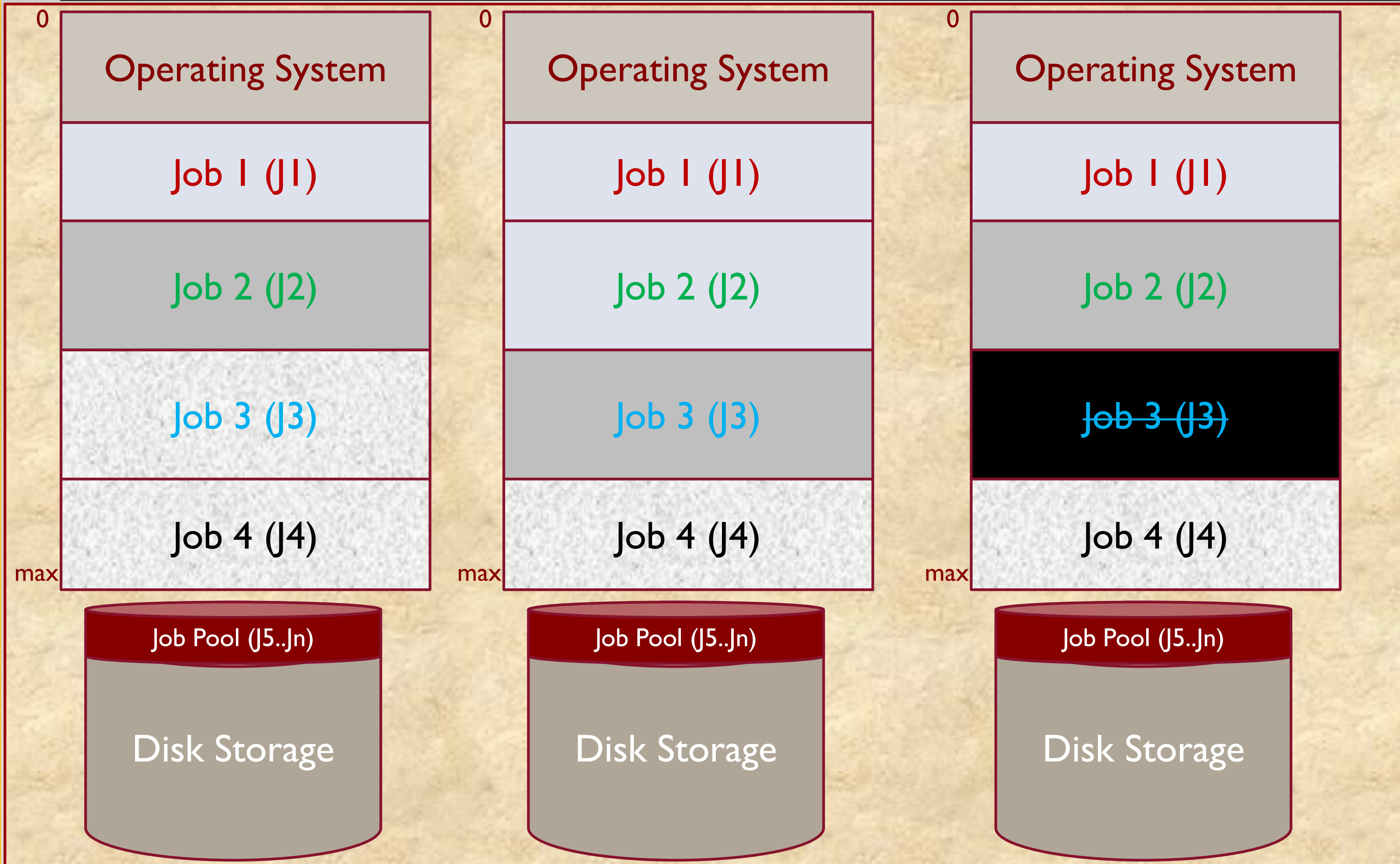
5. While the job ***(j1)*** is not making use of the CPU, the OS picks another job ***(j2)*** from memory and starts its execution.
6. If the job ***(j2)*** needs to wait while running, the OS switches to another job ***(j3)***.
7. When ***(j1)*** finishes waiting, it gets the CPU back after issuing an interrupt.

By this way, the CPU never stays idle as long as there is at least one job to be executed.

MULTIPROGRAMMING (3) – ILLUSTRATION (1)



MULTIPROGRAMMING (4) – ILLUSTRATION (2)



(4) J1 waits. J2 executes

(5) J1 & J2 wait.
J3 executes

(5) J2 executes
J1 waits, J3 completed.

MULTITASKING (I) – INTRODUCTION

In multiprogramming, a job keeps running until it completes execution, or it waits for something such as I/O. In the latter case, the OS activates another job.

However, the OS doesn't take user interaction into consideration.

In the previous example, the programmer running ***(J4)*** may have to wait too long until his job executes.

In other words, multiprogramming may not provide fair CPU assignment to the jobs allocated in the memory.

Multitasking (or time-sharing) is an extension of multiprogramming.

In multitasking, the OS switches quickly and frequently between multiple jobs (rather than waiting for the completion/suspension of the other jobs).

Thus, the OS gives the illusion to the user that the CPU is running his program all the time.

Since the user is present and interacting with the computer, the CPU response time to the user should be short.

Timesharing is used in the implementation of real-time applications.

MULTITASKING (2) – APPLICATIONS

Timesharing is used in the implementation of real-time applications.

Real-time applications attempt to supply a response within a certain bounded time period.

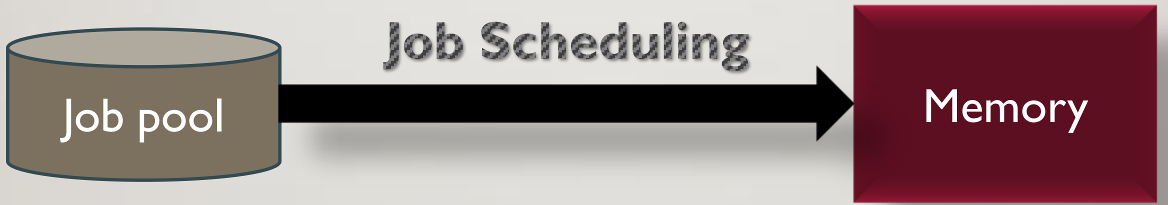
For example, a measurement from a petroleum refinery indicating that the temperature is too high. This might demand immediate attention to avoid an explosion.

The resources of a real-time application are often heavily underutilized: it is more important for such systems to respond quickly (short CPU response time) than it is for them to use their resources efficiently.

Scheduling

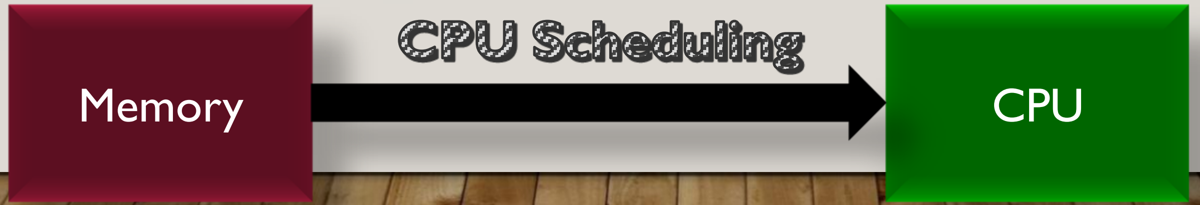
Both multiprogramming and multitasking require that several jobs be kept in memory simultaneously (at the same time).

If several jobs in the job pool (on disk) are ready to be brought into memory, and there is no enough room for all of them, then the OS must choose among them: this is Job Scheduling.



If several jobs in the job pool (on disk) are ready to be brought into memory, and there is no enough room for all of them, then the OS must choose among them: this is Job Scheduling.

The OS must also decide which job to run first (to assign to the CPU first): this is CPU Scheduling.



OPERATIONAL MODES (I)

Modern OS provide mainly two operational modes:

- Dual mode
- Multimode

Operational Modes (2) – Dual Mode (1)

For each type of interrupt, separate code segments in the OS determine the action to be taken. This is called System Call or Interrupt Service Routing or Interrupt Handler – as previously explained.

In a multiprogramming & multitasking systems, an OS must be properly designed by ensuring that an incorrect or malicious program cannot cause other programs to execute incorrectly.

In a multiprogramming & multitasking systems, an OS is designed with two main objectives:

1. Protect the OS from being accessed by the system users.
2. Protect a user code from being accessed by other users.

In order to implement this, an OS is designed with two execution modes:

- The kernel mode in which the execution of the OS routines are executed.
- The user mode in which the user-defined codes are executed.

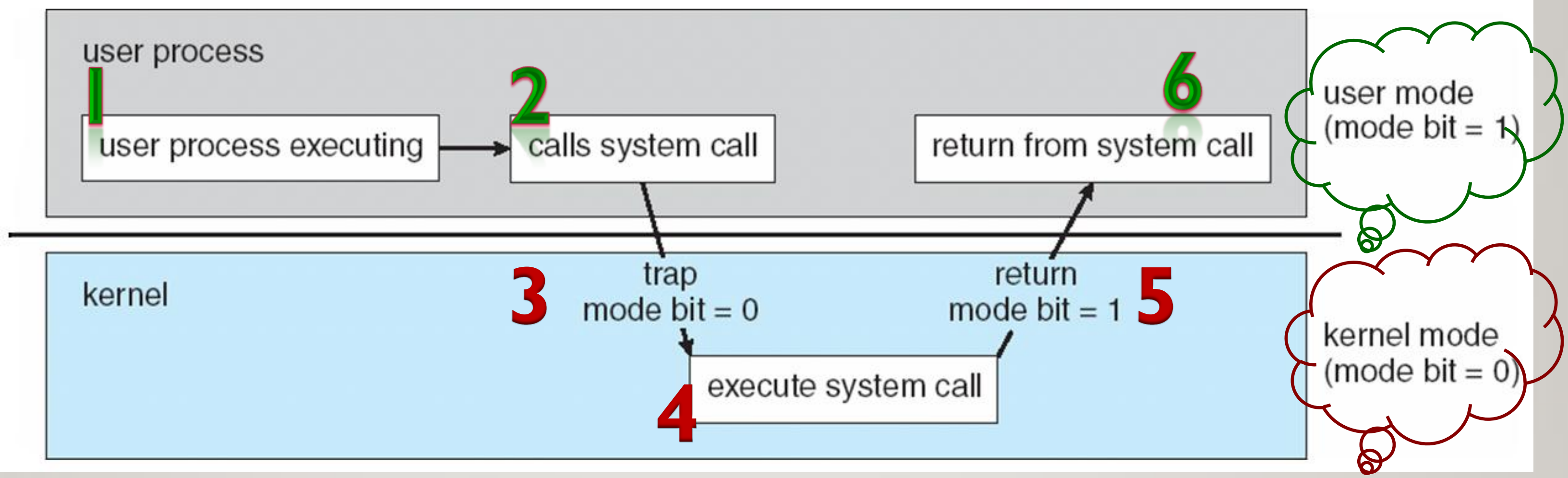
OPERATIONAL MODES (3) – DUAL MODE (2) – THE MODE BIT (1)

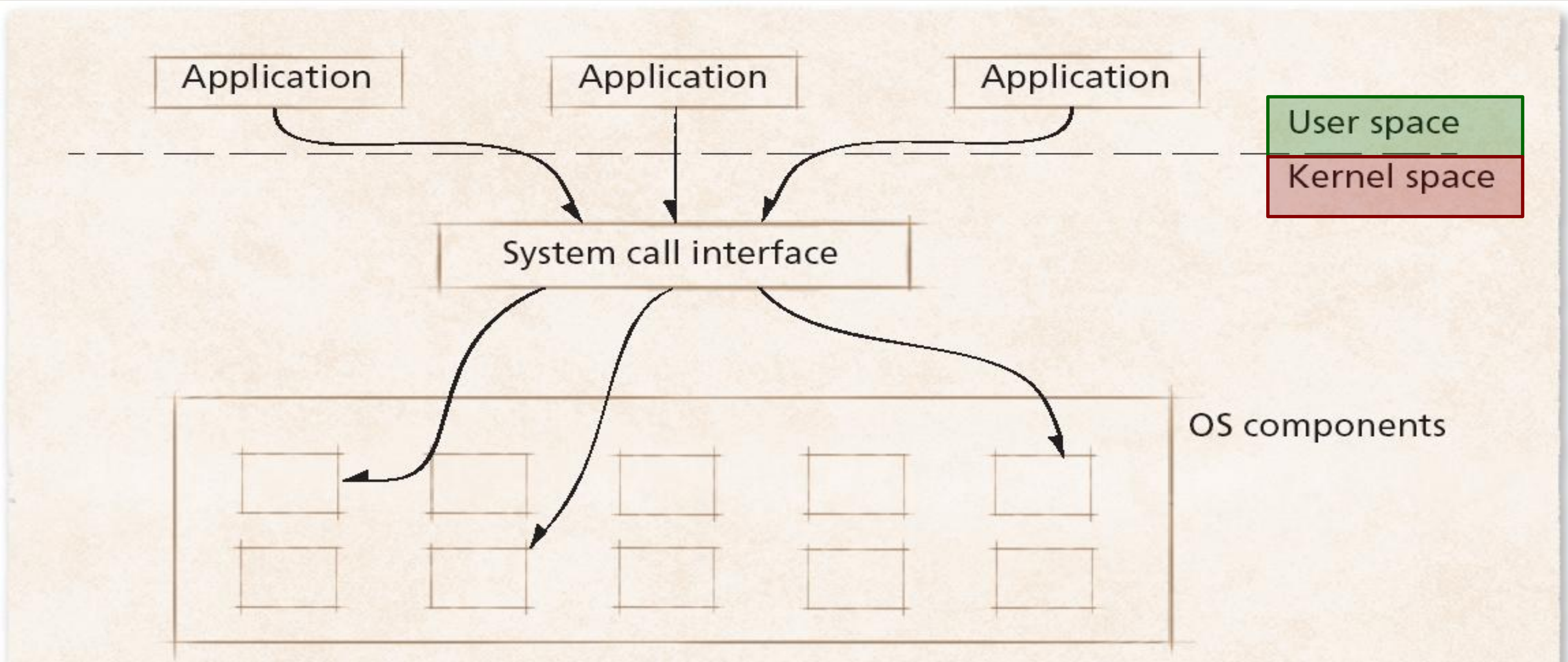
The kernel mode is also called supervisor mode or system mode or privileged mode

The approach taken by most computer systems is to provide hardware support that allows us to differentiate between various modes of execution as follows:

- A bit, called the mode bit, is added to the computer hardware.
- If the mode bit is set to 0, then the computer is in the kernel mode.
- If the mode bit is set to 1, then the computer is in the user mode.
- At system boot time, the hardware starts in the kernel mode (mode bit = 0)
- The OS is then loaded and starts executing the users applications in the user mode (mode bit = 1)
- Whenever a trap/interrupt occurs, the hardware switches from the user mode to the kernel mode (mode bit = 0) and calls the interrupt service routine (ISR).
- When the interrupt service routine (interrupt handler) completes execution, the system switches to the user mode before passing control to the user program (mode bit = 1)

OPERATIONAL MODES (4) – DUAL MODE (3) – THE MODE BIT (2)





OS FEATURES

Operational Modes (5) – Dual Mode (4) – Privileged Instructions

Privileged instructions may cause harm if used improperly.

Therefore, privileged instructions may execute only in the kernel mode.

Examples of privileged instructions include:

- Setting the mode bit to 0 intentionally (ie. Kernel mode).
- Manipulation of memory protection keys associated with each process (to be covered later).
- Setting the computer clock.
- Interrupt management

So, the hardware allows privileged instructions to execute only in the kernel mode.

If an attempt is made to execute a privileged instruction in the user mode, the hardware does not execute it, and triggers (causes) a trap to the OS.

Operational Modes (6) – Multimode

Virtualization is a technique that allows the user to have multiple OSs on a single physical machine.

Each installed OS is called a Virtual Machine (VM).

The software that manages the virtual machines is known as Virtual Machine Manager (VMM) or Hypervisor.

The main role of a VMM is to control the virtual machines installed on the top of the physical machine.

The VMM should be assigned more privileges than the user process, but less than the OS.

Therefore, we need more than two modes → more than a mode bit is needed.

With two bits to designate the operational mode, we have up to four modes (00, 01, 10, 11)

Operational Modes (7) – OS Examples

Microsoft Disk Operating System (MS-DOS) was designed for the Intel 8088 architecture in 1980s.

Intel 8088 was designed with no mode bit. Therefore, it does not support the dual mode.

This entails that a user program may inadvertently (or intentionally) overwrite the OS with data.

However, modern versions of the Intel CPU are provided with a mode bit.

Therefore, all modern OSs such as Microsoft Windows, Unix, and Linux take advantage of this feature (hardware piece) to provide greater protection to the OS.

Timers (1)

We must ensure that the OS always maintains control over the CPU.

If a user program enters into an infinite loop, or fails to perform an interrupt, then control would never returned to the OS.

The timer ensures that control returns to the OS in spite of ANY error encountered in the user's program.

Two types of timers are designed to be used in a computer system:

- Fixed timers
- Variable timers

TIMERS (2) – FIXED TIMERS

Fixed timers are set to send an interrupt to the OS after a specific period of time, say 1/60 second.

The following include some examples in which a fixed timer send an interrupt after 100 mseconds (milli seconds) [1 second = 1000 milli-seconds]

- An infinite loop in a running code
  - Every 100 msec, the OS assigns the CPU to read the keyboard input looking for an ESC key to quit the program.
  - Without such interrupt, the CPU would execute the infinite loop forever, and you won't be able to enter the ESC key to stop the running program.
- Downloading a file from a slow internet connection
  - After 100 msec, the clock sends an interrupt to the CPU to stop his trials in downloading the file. This would call the relevant Interrupt Handler.
  - The OS also interacts with the user by giving him a message indicating that "the file is not found", or "there is no Internet connection".
  - Without such interrupt, the CPU would spend a very long time waiting for the file to be downloaded. Therefore, affecting the CPU resource utilization negatively.

TIMERS (3) – VARIABLE TIMERS (1)

Variable timers are generally implemented with a fixed-rate clock and a counter.

Variable timers work as follows:

- The OS sets the counter to a specific value.
- Every time the clock ticks (every clock cycle), the counter is decremented by 1.
- When a counter reaches the value 0, the clock sends an interrupt to the OS.

Example:

- Assume that the hardware is provided with a 10-bit counter.
- This means that the OS may set the counter at a value up to $2^{10} = 1{,}024$.
- Assume also that the clock ticks every 1 millisecond.
- Therefore, an interrupt may be issued at intervals from 1 ms to 1,024 ms in steps of 1 ms.

Instructions that set the timer are classified as privileged instructions.

TIMERS (4) – VARIABLE TIMERS (2)

Consider the following practical example:

- Assume that the time allowed to download a file from the Internet is 7 minutes.
- 7 is represented as 111 in binary.
- Assume also, that the clock cycle is 1 second for simplicity.
- The counter is therefore initialized to 7x60 seconds = 420 seconds.
- Every second, the timer interrupts, and decrements the counter by 1.
- If the counter is positive, control is returned to the user program (downloading continues).
- If the counter is negative, the OS terminates the program (downloading) and sends a message to the user.

H/W Interrupts – Example

A time-out is set to a specific time. When the time elapses the clock sends an interrupt.