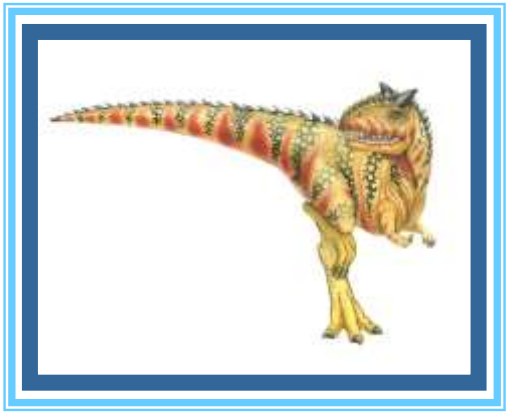
Chapter 1: Introduction



Course References

- No mandatory textbook
  - Lecture slides are your primary reference
  - Other references to be provided as we proceed
  - Recommended textbook:

https://codex.cs.yale.edu/avi/os-book/OS10/index.html

Operating System Concepts
Tenth Edition

Avi Silberschatz
Peter Baer Galvin
Greg Gagne

John Wiley & Sons, Inc.
ISBN 978-1-118-06333-0

Face The Real World of Operating Systems Fully Equipped

Workload

- 1 midterm exam
- 2 take-home assignments
- 1 final exam

- 6 lab assignments

What to expect

- The course would be quite involved
  - Lot of concepts
  - Some programming
  - Reasonable workload
- Let me know if you have any question
- I guarantee that
  - I will encourage you to do your best
  - You'll have fun
  - I'll help you learn as much as I can – don't hesitate to ask for help whenever needed
  - Although you won't become experts, you will learn enough to move on!

What I expect of you

- Please do attend each Lecture and each Lab!
- Review lecture slides after each lecture!
- Ask questions during lectures!
- Ask questions over **email**
  - hzb564@jnu.edu.cn this one!
- Try to start early on assignments
  - Don't wait until the very last minute!
- Follow the instructions and submit assignments on time

Attendance Policy

- Attendance will be taken randomly

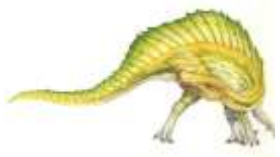
- If you fail to attend the lectures at **3 times** without prior notification of your absence, your attendance grade will be **0**.

Late Homework Policy

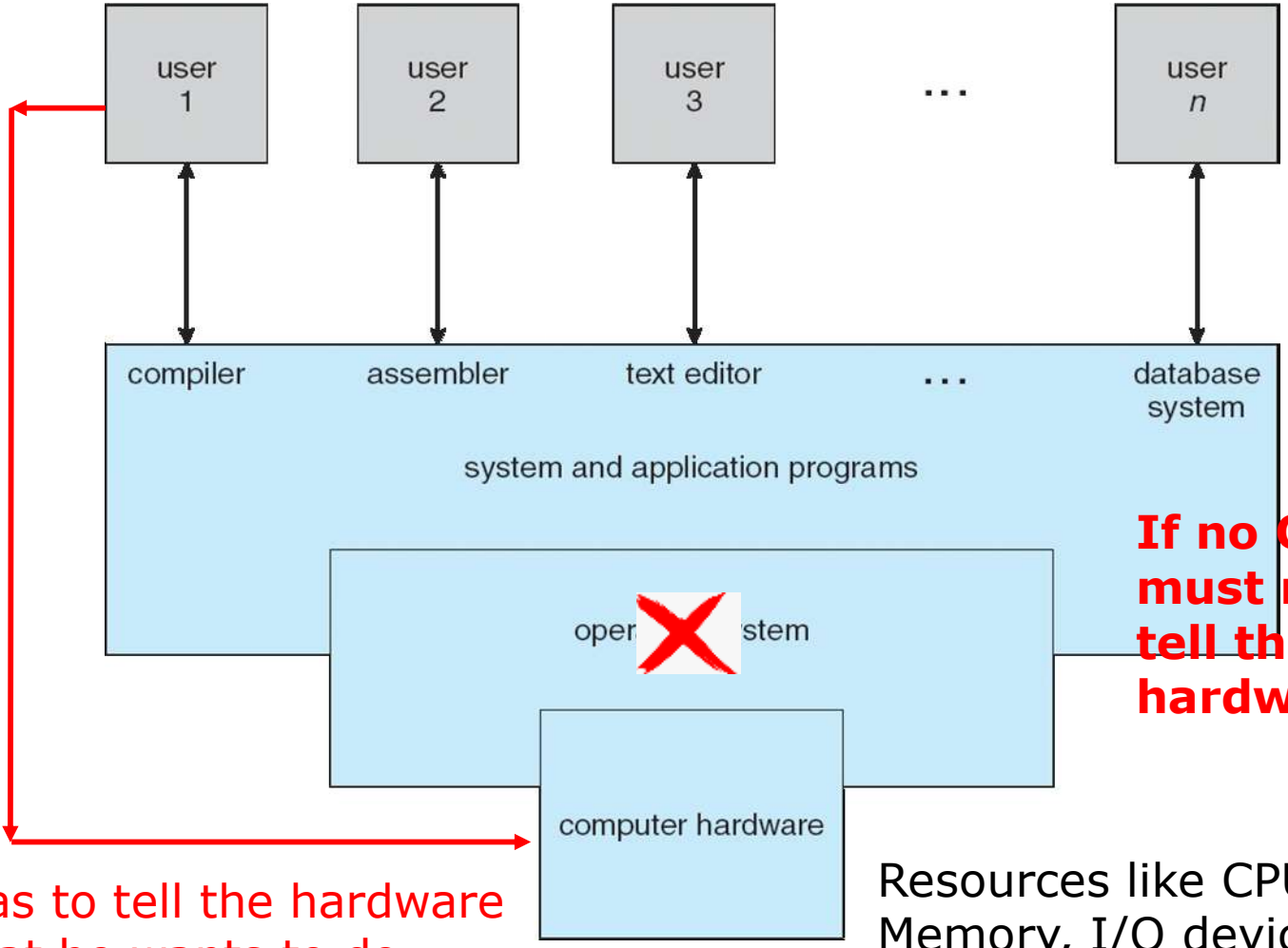
- None – **no late homework is allowed**
- Either you submit on time and your homework will be graded OR you submit late, and the homework is NOT graded
- You should stick to deadlines
- Exception will be made ONLY under genuine circumstances

What is an Operating System?

- An Operating System (OS) is a **program** that manages the computer **hardware**
- It also provides a **basis** for **Application Programs**
- A program that acts as an **intermediary** between computer **user** and computer **hardware**



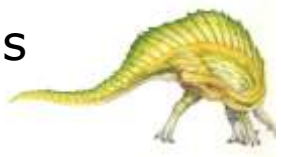
Four Components of a Computer System



If no OS, you must manually tell the computer hardware!

Has to tell the hardware that he wants to do something!

Resources like CPU, Memory, I/O devices



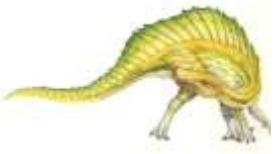
What is an Operating System?

- Operating system goals:
  - Execute user programs and make solving user problems **easier**
  - Make the computer system **convenient** to use
  - Use the computer hardware in an **efficient** manner

Convenience

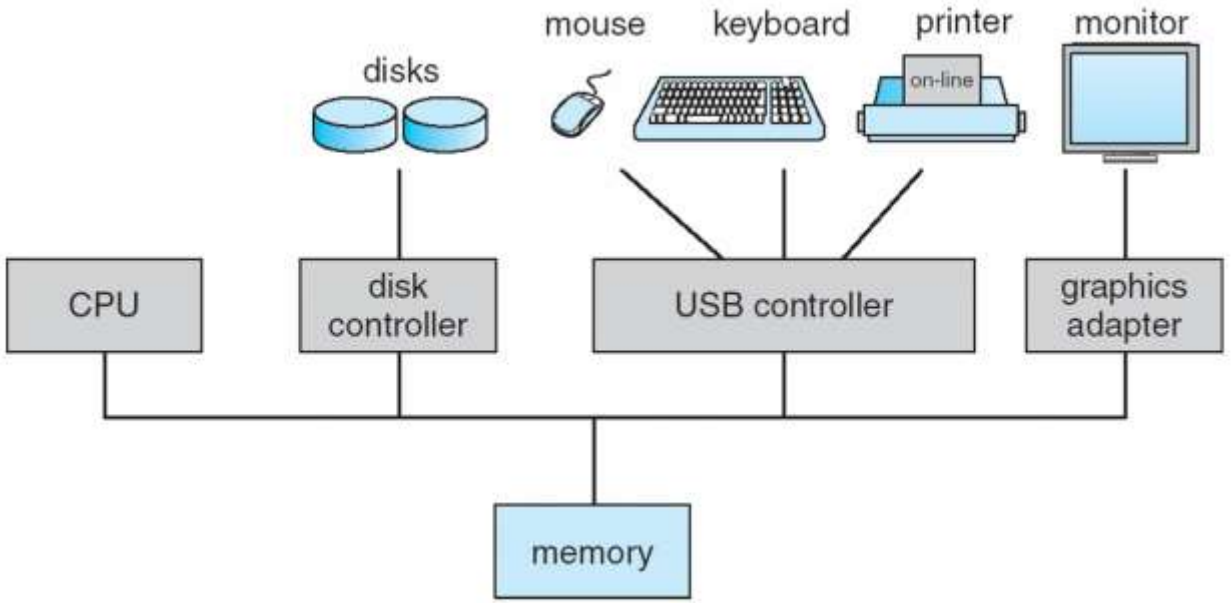
Efficiency

Security

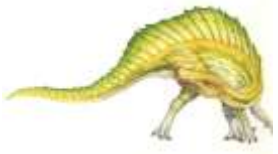


Computer System Organization

- One or more **CPUs**, **device controllers** connect through common **bus** providing access to **shared memory**
  - Each device controller is in charge of a particular device type

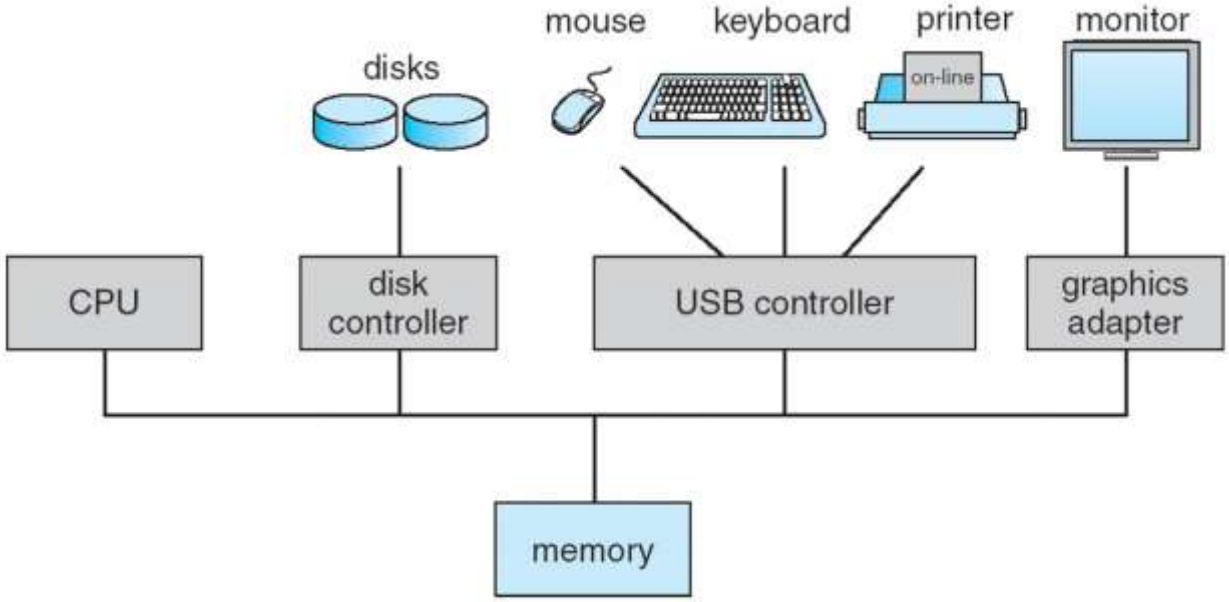


Buffer

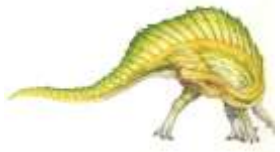


Computer System Organization

- One or more **CPUs**, **device controllers** connect through common **bus** providing access to **shared memory**
  - Each device controller is in charge of a particular device type
  - The CPU and the device controllers can **execute concurrently**, competing for memory cycles
  - To ensure **orderly access** to the shared memory, a **memory controller** is provided whose function is to **synchronize access** to the memory

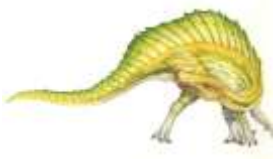


Buffer



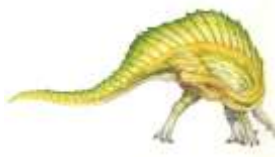
What Operating Systems Do — User view

- One PC user want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles



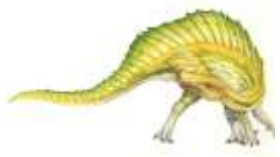
What Operating Systems Do — System view

- OS is a **resource allocator**
  - Manages all resources
    - Example of resources?
  - Decides between conflicting requests for efficient and fair resource use
    - Examples of conflicting requests?
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer, especially for I/O
    - Examples of improper use of computer?



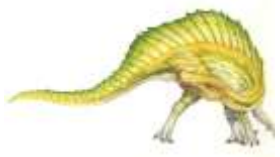
Computer Startup

- Where is the OS stored on the computer?
- What operation or program starts the OS?
- **bootstrap program** is loaded at power-up or reboot
  - Loads operating system kernel and starts execution
    - Locate the operating-system kernel and load it into memory
  - Some services are provided outside of the kernel, by **system software**
  - Typically stored in ROM, generally known as **firmware**

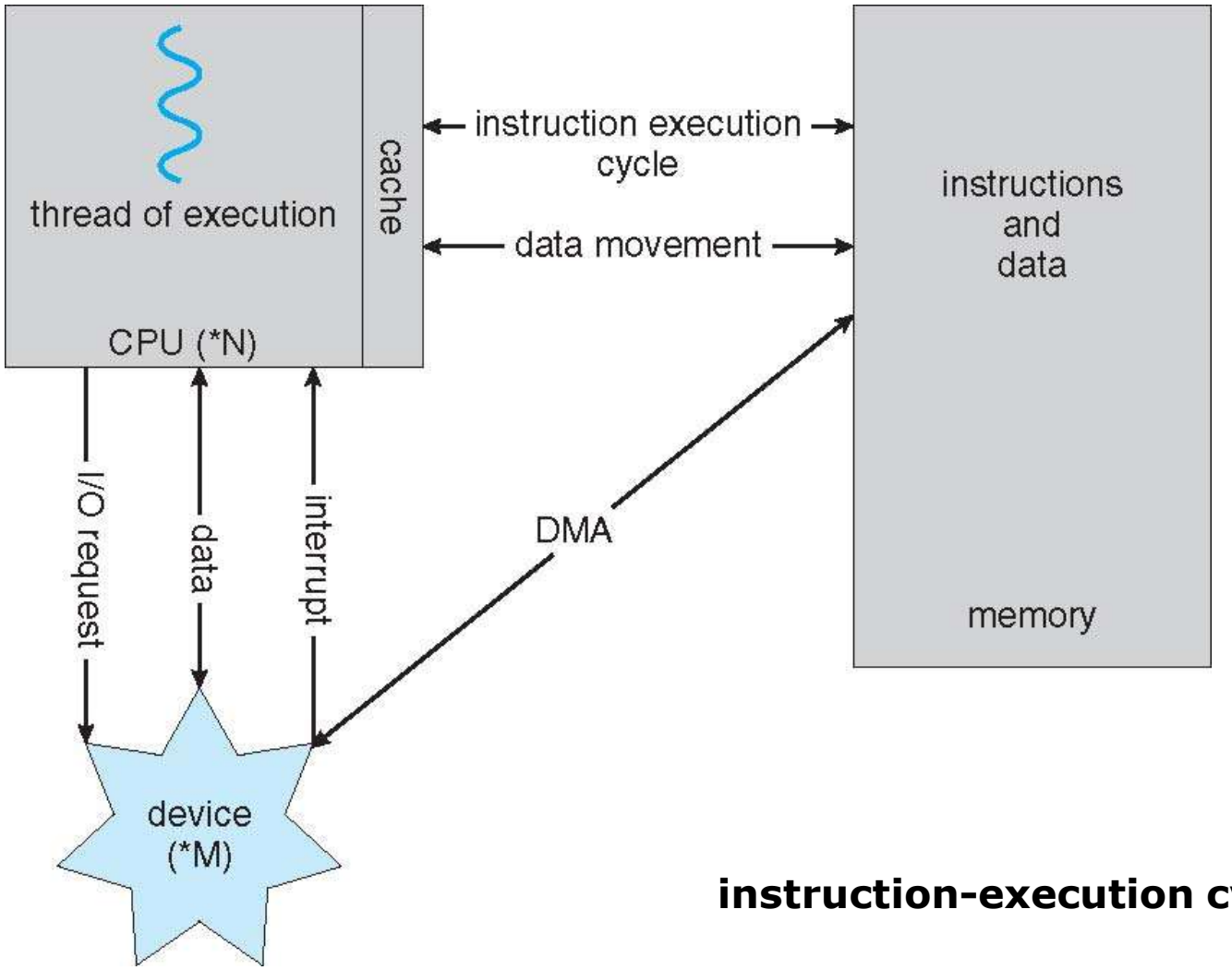


Storage Structure

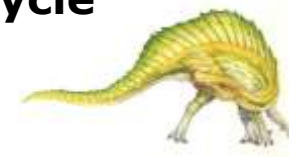
- Main memory – only large storage media that the CPU can access directly
  - **Also called Random access memory (RAM)**
  - **Rewritable**
- Computers use other forms of memory as well
  - Read-only memory (ROM)
- Interaction is achieved through a sequence of **load** or **store** instructions to specific memory addresses



How a Modern Computer Works

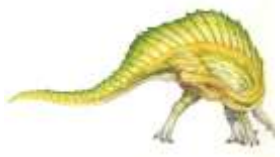


instruction-execution cycle



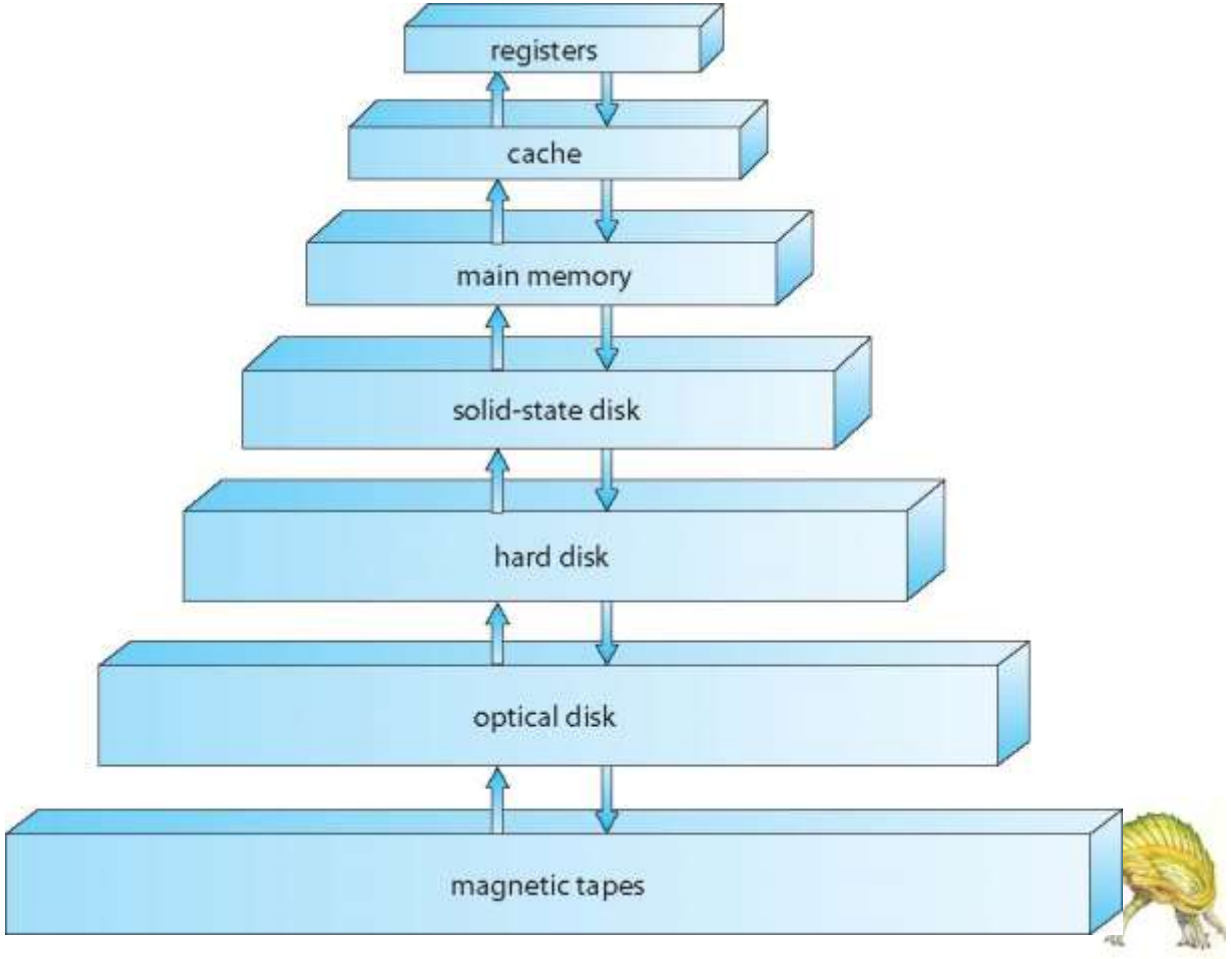
Storage Structure (Cont.)

- Ideally, we want the programs and data to reside in main memory permanently. Usually, it is not possible
  - Too small
  - **Volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
  - Hard disks – the most common secondary-storage device
  - Solid-state disks – faster than hard disks, nonvolatile
    - Stores data in a large DRAM, a hidden hard disk, a battery
    - Becoming more popular: flash memory
  - **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage



Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Size
  - Volatility



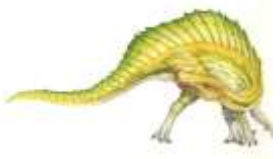
Programmed I/O

How CPU gets a character from the keyboard?

Programmed I/O

Interrupted I/O

Direct Memory Access (DMA)



Programmed I/O

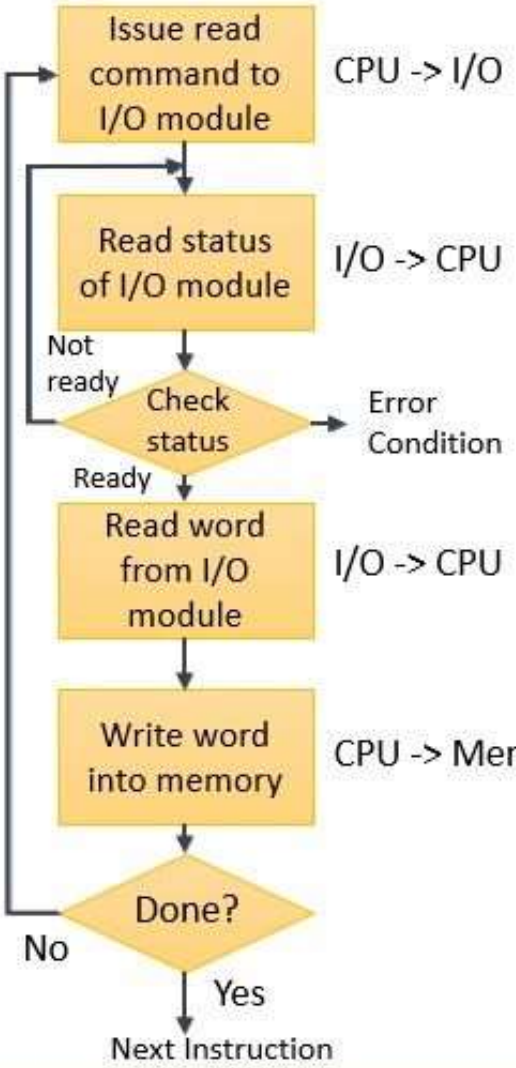
How CPU gets a character from the ke

Programmed I/O: a technique that we
between the processor and the I/O modul

Regulated with the help of a program.

What's the greatest drawback of programmed I/O?

a processor keeps on checking whether the I/O module is ready for reception and transmission of data or whether the I/O module has completed the desired task or not. This long waiting of the processor deteriorates the performance of the system.



Programmed I/O to transfer data from I/O module to memory

Interrupted I/O

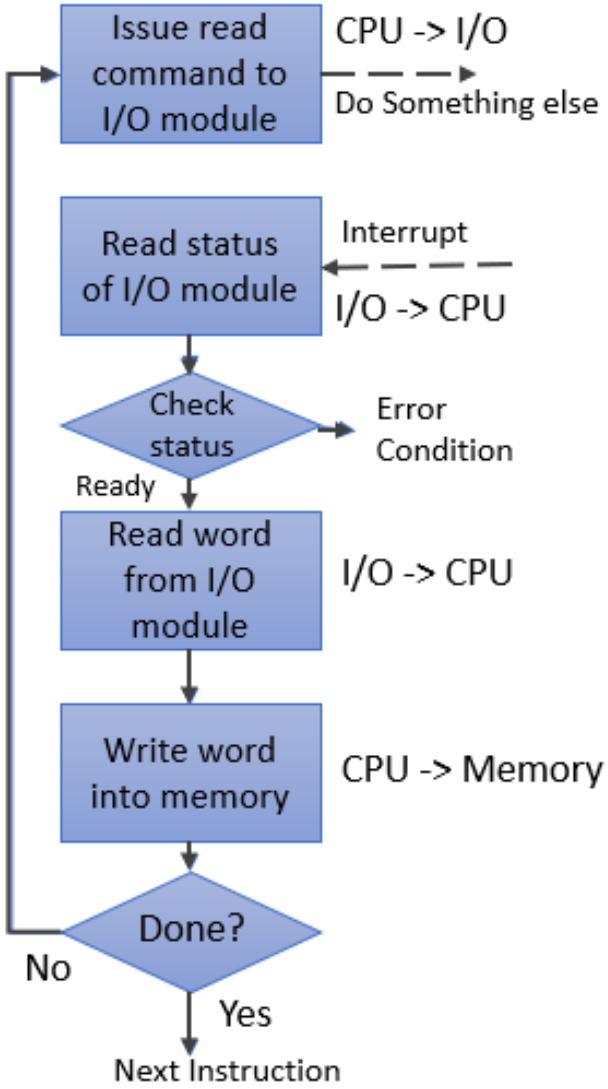
How CPU gets a character from the k

Interrupted I/O: an approach to trans
and 'I/O devices' through the 'processor'

Involves the use of **interrupt** to exchang
memory.

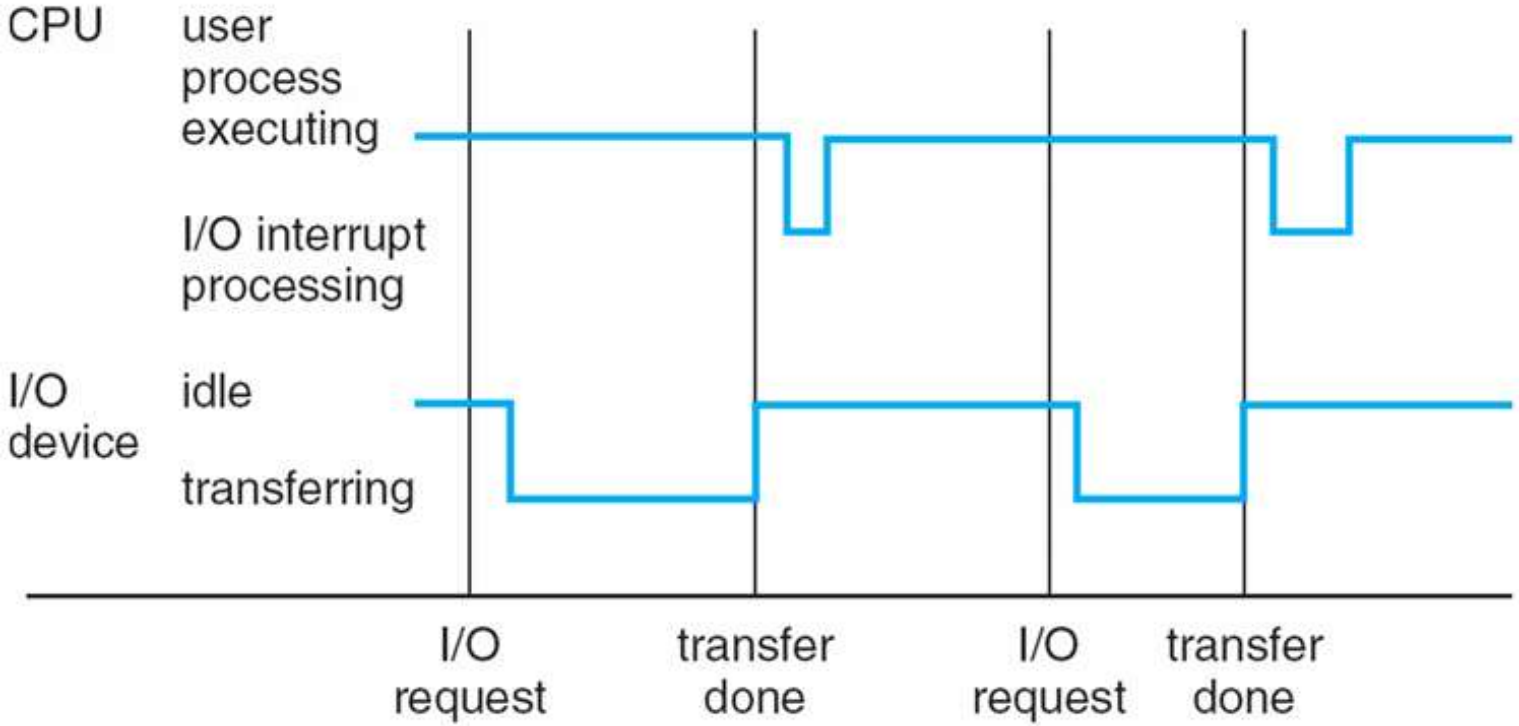
Main idea: after issuing the I/O comman
processor can get itself busy doing some

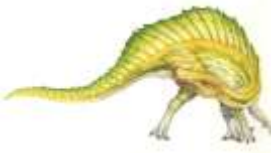
issues an interrupt signal to the processor



Interrupted I/O to Transfer Data from I/O Module to Memory

Interrupted I/O

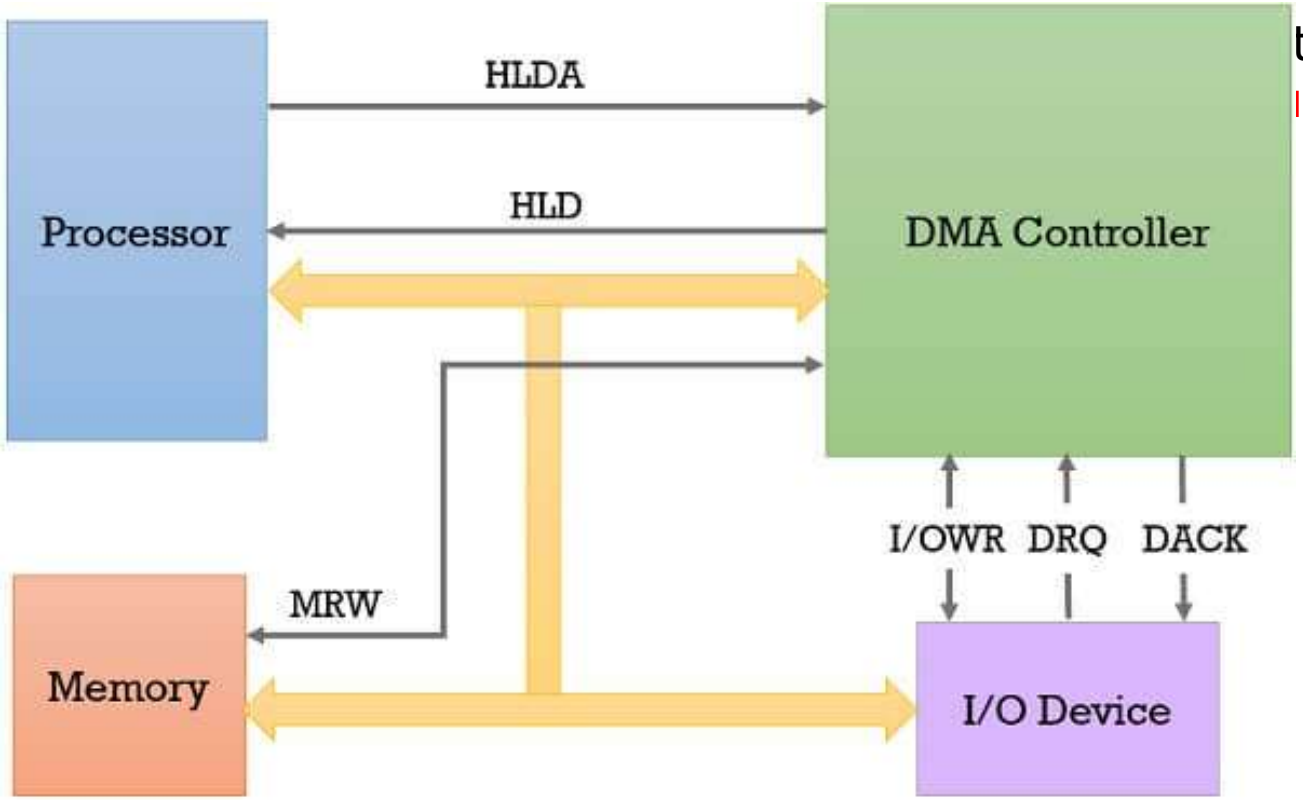




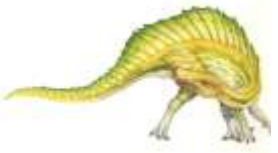
Direct Memory Access (DMA)

How CPU gets a character from the keyboard?

Direct ta
betwee it the
partici

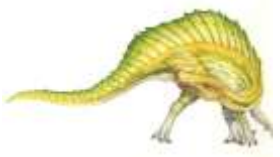


DMA Controller Data Transfer



Direct Memory Access (DMA)

- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte
- Used for high-speed I/O devices able to transmit information at close to memory speeds



Direct Memory Access (DMA)

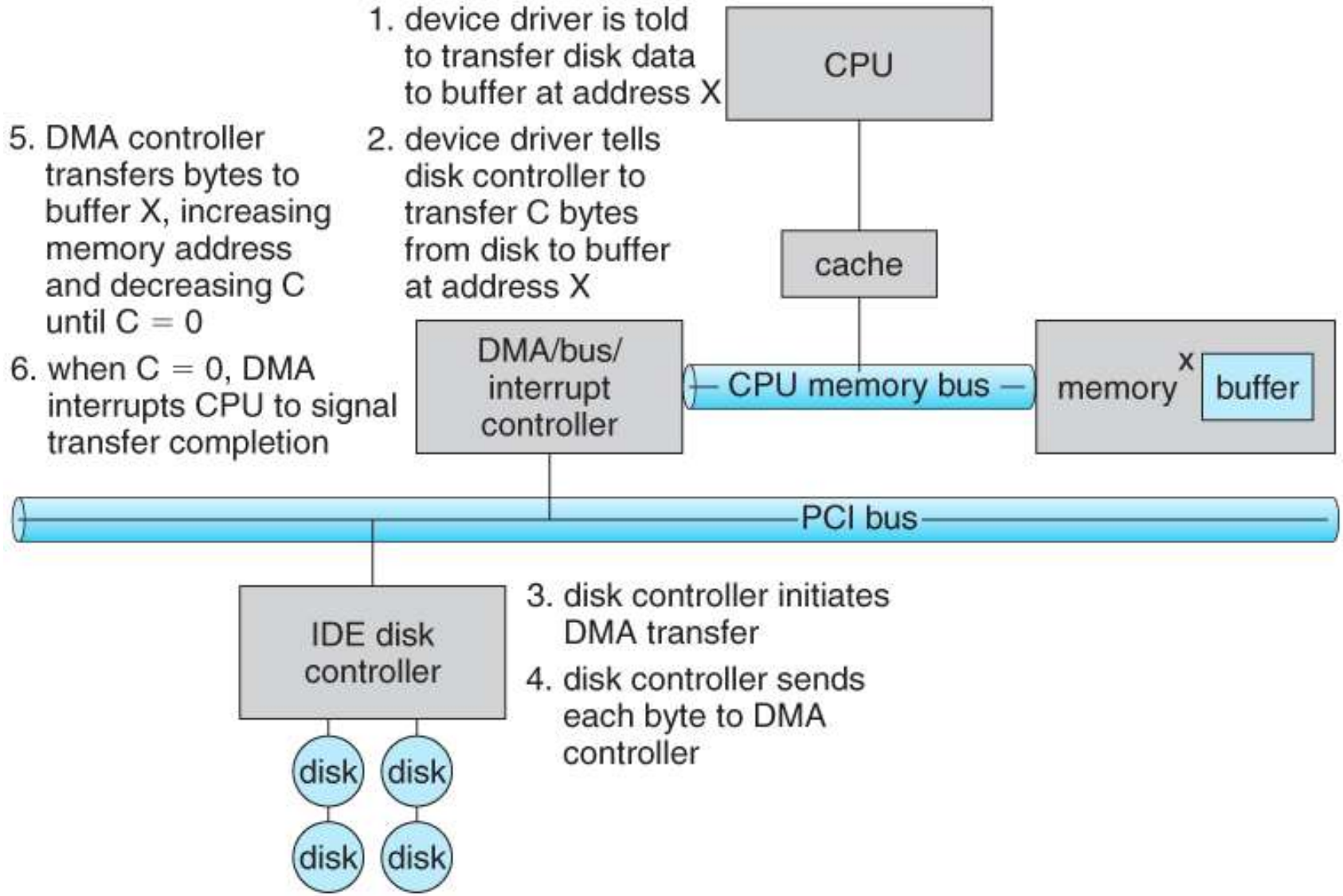


Figure 13.5 - Steps in a DMA transfer.

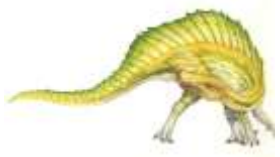
Direct Memory Access (DMA)

The DMA controller transfers the data in three modes:

1.Burst Mode: Here, once the DMA controller gains the charge of the system bus, then it releases the system bus only after completion of data transfer. Till then the CPU has to wait for the system buses.

2.Cycle Stealing Mode: In this mode, the DMA controller forces the CPU to stop its operation and relinquish the control over the bus for a short term to DMA controller. After the transfer of every byte, the DMA controller releases the bus and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte.

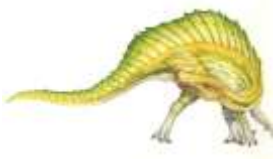
3.Transparent Mode: Here, the DMA controller takes the charge of system bus only if the processor does not require the system bus.



Direct Memory Access (DMA)

When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished, and the processor can take control over the bus again and start processing where it has left.

Different between DMA and interrupted I/O?



Direct Memory Access (DMA)

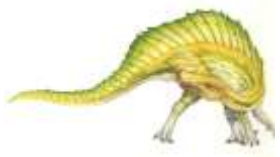
Direct Memory Access Advantages and Disadvantages

Advantages:

1. Transferring the data without the involvement of the processor will speed up the read-write task.
2. DMA reduces the clock cycle requires to read or write a block of data.
3. Implementing DMA also reduces the overhead of the processor.

Disadvantages

1. As it is a hardware unit, it would cost to implement a DMA controller in the system.



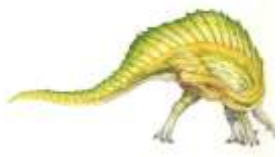
Comparison

Programmed I/O: It transfers data at a high rate, but it can't get involved in any other activity during data transfer.

Interrupted I/O: the processor doesn't keep scanning for I/O devices ready for data transfer. But, it is fully involved in the data transfer process.

The above two modes of data transfer are not useful for transferring a large block of data.

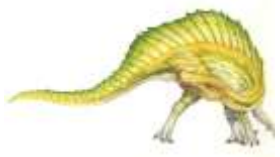
Direct Memory Access (DMA) : completes this task at a faster rate and is also effective for transfer of large data block.



Operating System Structure

- **Multiprogramming** is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute

- Basic idea of multiprogramming
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job



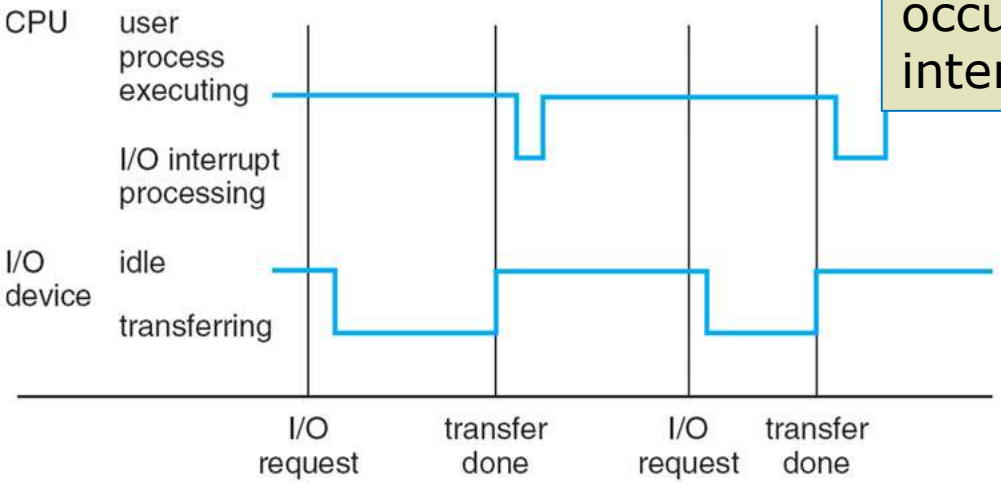
Operating System Structure (Cont.)

- **Timesharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can **interact** with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
- Timesharing and multiprogramming require that several jobs be kept simultaneously in memory
  - If several jobs ready to be brought into memory ⇨ **Job scheduling**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
- In timesharing, the OS must ensure reasonable response time
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

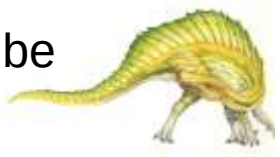
Interrupts

- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
    - I/O operation is completed

Events are almost always signaled by the occurrence of an interrupt or a trap



  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Invalid memory access
    - A specific request from user program that an OS service be performed

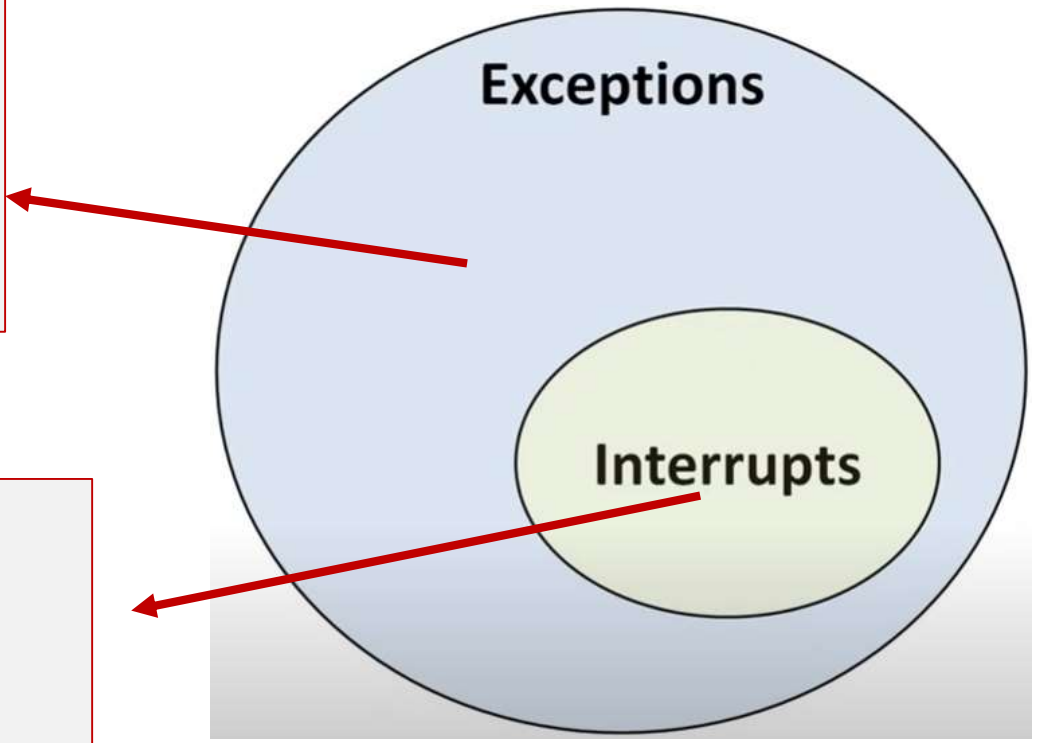


Interrupts (Cont.)

- Exceptions
  - Any event that can alter normal CPU instruction execution flow

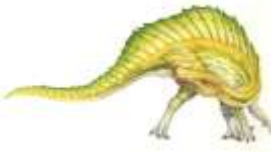
"**Internal**" events that are **abnormal**:

e.g.:
Invalid Instruction
Illegal address access



"**Hardware**" driven signals:

e.g.:
External Signals
Peripheral Flags

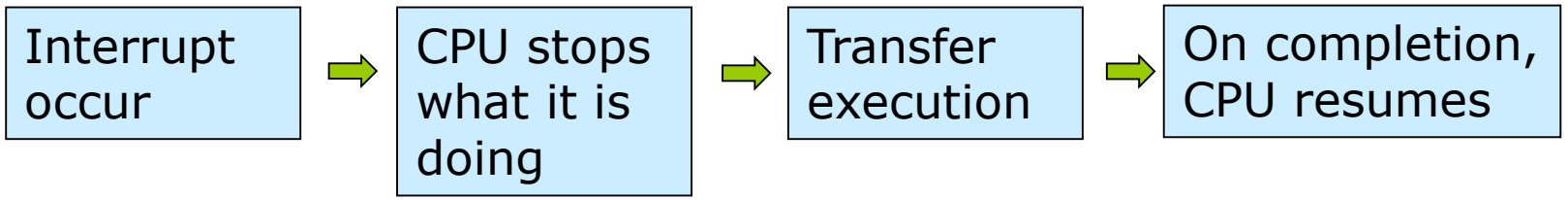


Interrupts (Cont.)

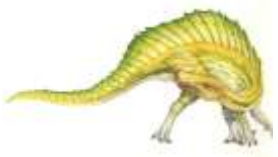
Given a specific interrupt, what action should be taken?

Separate **segments of code** in the operating system determines what action should be taken

- An **interrupt service routine** is provided to deal with the interrupt

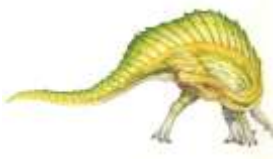


How to find the corresponding interrupt service routine



Interrupts (Cont.)

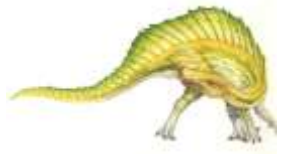
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains **the addresses of all the service routines**
  - A table of pointers to interrupt routines



Interrupts (Cont.)

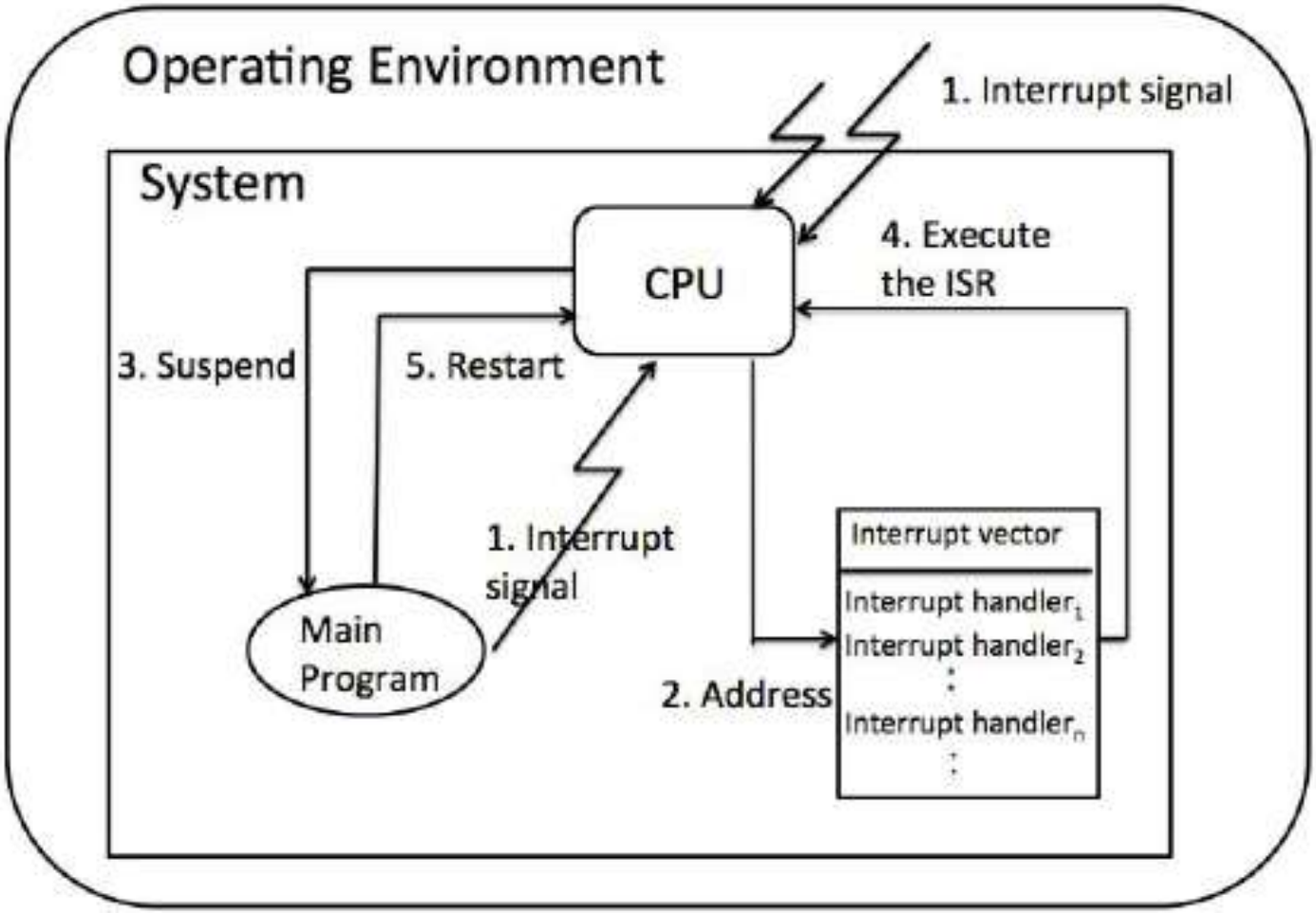
- Every interrupt type is assigned a number:
  - Interrupt vector
- A table of pointers to interrupt routine
- When an interrupt occurs, the vector determines what code is invoked to handle the interrupt
  - Example: vector 14 page fault handler

0	Divide Error
2	Non-Maskable Interrupt
3	Breakpoint Exception
6	Invalid Opcode
11	Segment Not Present
12	Stack-Segment Fault
13	General Protection Fault
14	Page Fault
18	Machine Check
32-255	User Defined Interrupts



Interrupts (Cont.)

- The interrupt must transfer control to the appropriate interrupt service routine
- Thus, the interrupt routine is called indirectly through the table

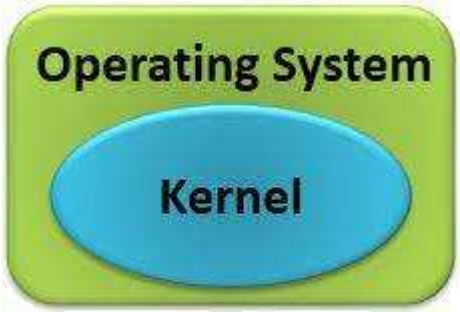


System Call

User program

Request service

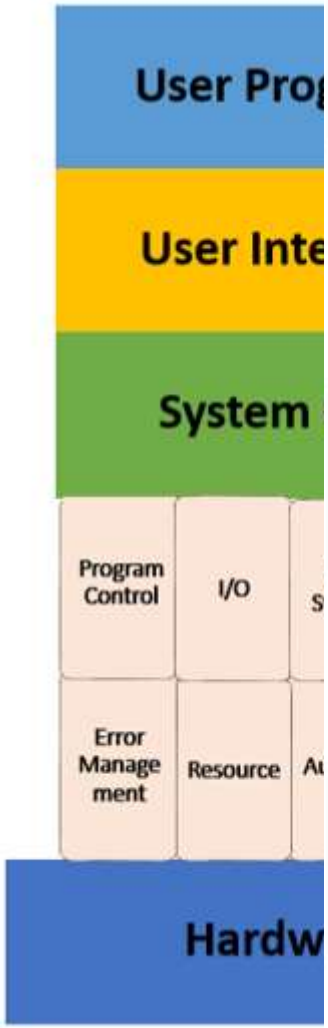
e.g.:
open a file
create a process
……



API: open(), fork()
……

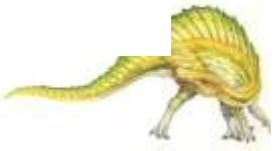
- **System call provides** the services of the operating system to the user programs via Application Program Interface (API).
- It provides an **interface** between a process and operating system to allow user-level processes to request services of the operating system.
- System calls are the only entry points into the kernel system.

System Call



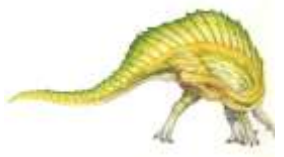
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

in OS



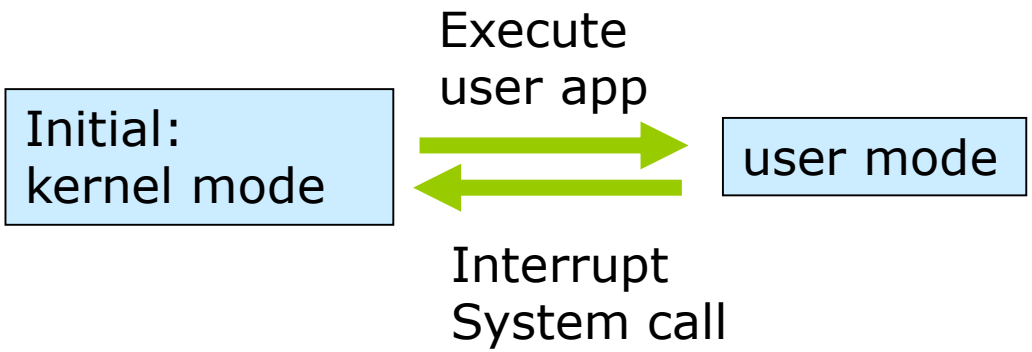
Operating-System Operations (cont.)

- Proper execution: distinguish between the execution of operating-system code and user-defined code
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
  - **System call** changes mode to kernel, return from call resets it to user
- Protection operating system
  - Some instructions designated as **privileged**, only executable in kernel mode

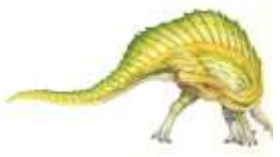


Operating-System Operations (cont.)

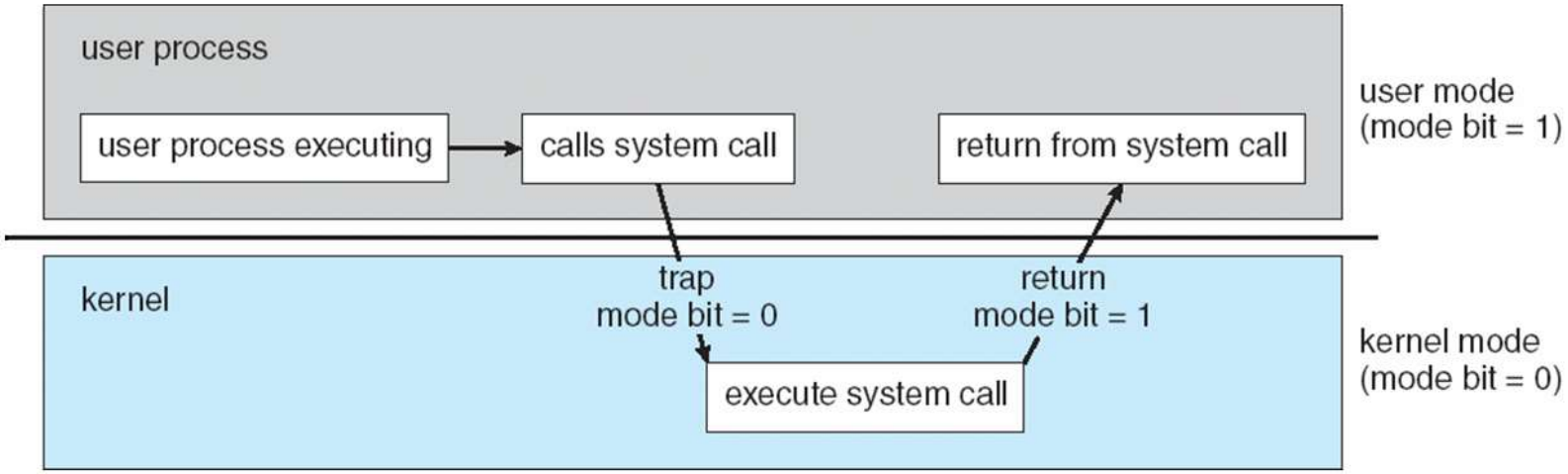
- Life cycle of instruction execution



Without hardware-supported dual mode (for example, MS-DOS), a user program can wipe out the operating system by writing over it with data; multiple programs are able to write to a device at the same time

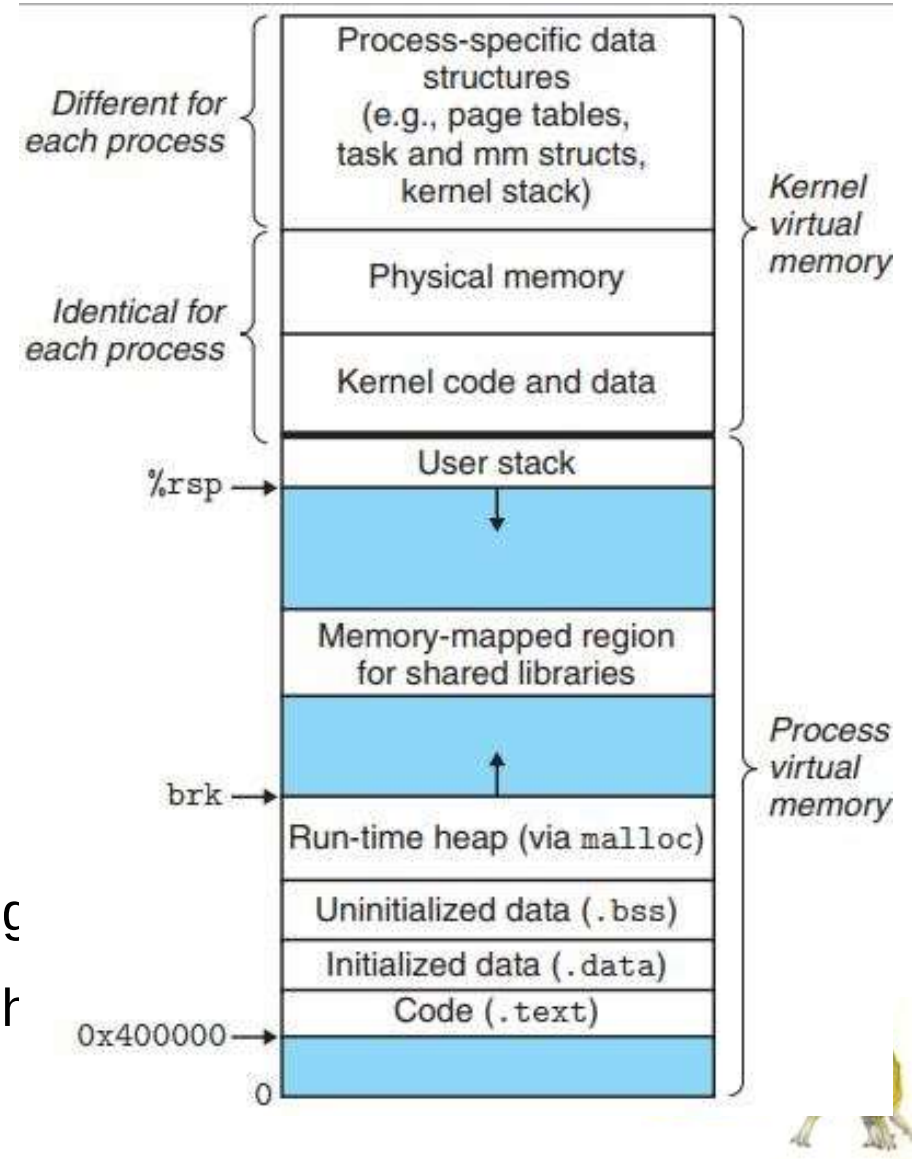


Transition from User to Kernel Mode



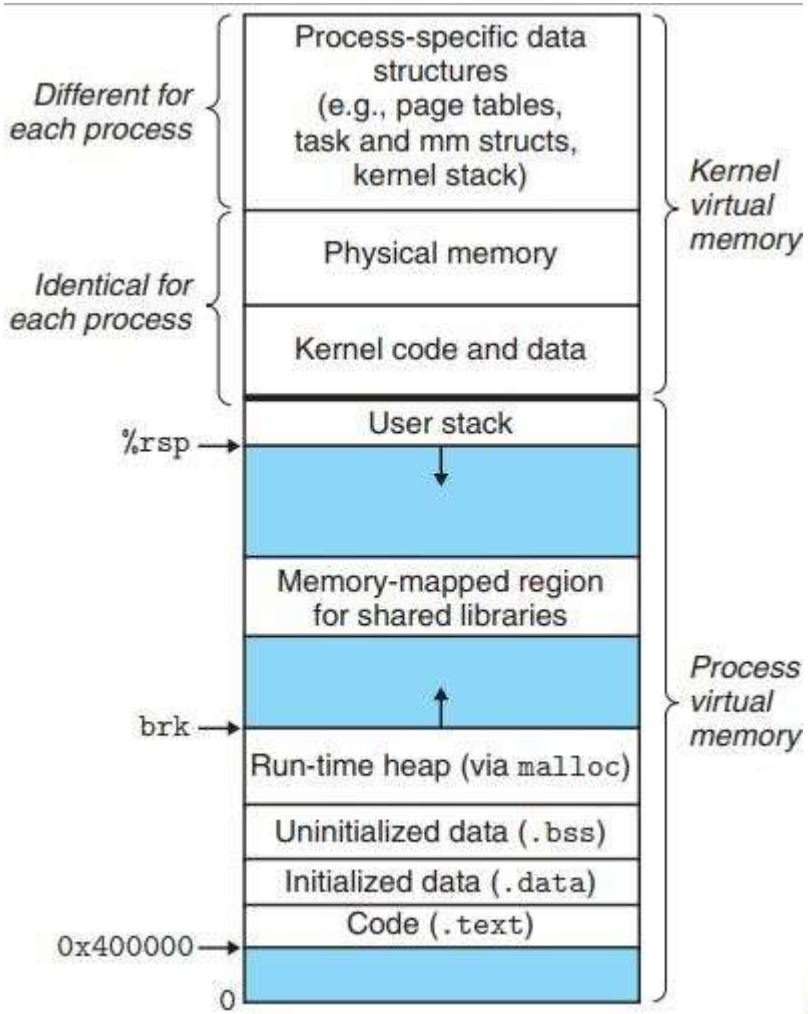
User mode

- Because an application's virtual address space is private, one application cannot alter data that belongs to another application
- Each application runs in isolation, and if an application crashes, the crash is limited to that one application.
- Other applications and the operating system are not affected by the crash



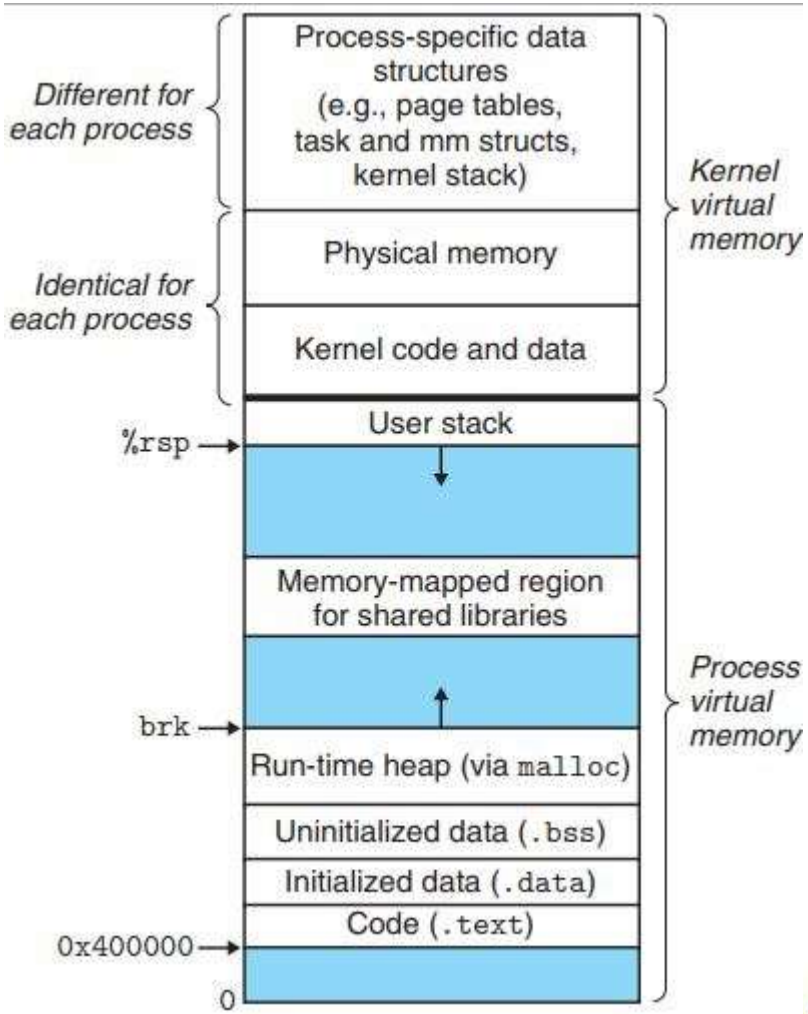
User mode

- In addition to being private, the virtual address space of a user-mode application is limited
- A processor running in user mode cannot access virtual addresses that are reserved for the operating system.
- Limiting the virtual address space of a user-mode application prevents the application from altering, and possibly damaging, critical operating system data.



Kernel mode

- All code that runs in kernel mode shares a single virtual address space.
- This means that a kernel-mode driver is not isolated from other drivers and the operating system itself.
- If a kernel-mode driver accidentally writes to the wrong virtual address, data that belongs to the operating system or another driver could be compromised.
- If a kernel-mode driver crashes, the entire operating system crashes.



Quiz

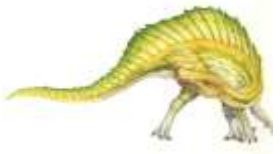
Which of the following should NOT be allowed in user mode?

a) Disable all interrupts.

b) Read the time-of-day clock

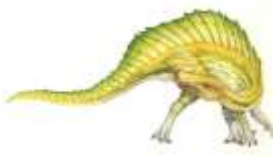
c) Set the time-of-day clock

d) Perform a trap



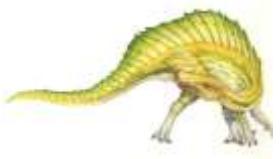
Process Management Activities

- The operating system is responsible for the following activities in connection with process management:
  - Scheduling processes and threads on the CPUs
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling



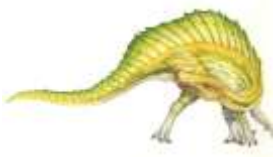
Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed



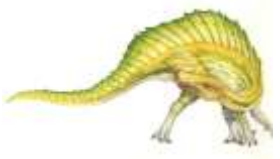
Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - **File**
    - a collection of related information defined by its creator
    - files represent programs and data



Mass-Storage Management

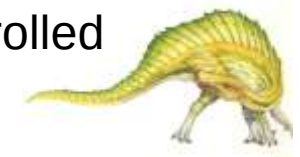
- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling



Performance of Various Levels of Storage

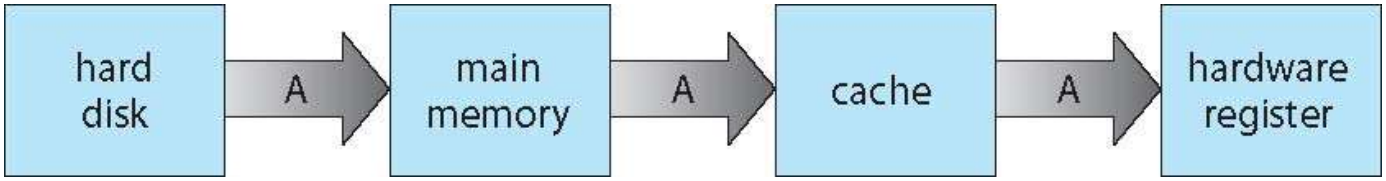
Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

- Figure compares performance of various levels of storage
- For example, data transfer from cache to CPU and registers is usually a hardware function, with no operating system intervention
- In contrast, transfer of data from disk to memory is usually controlled by the operating system

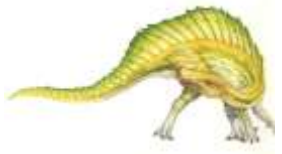


Migration of data “A” from Disk to Register

- Multitasking environments must **be careful** to use most recent value, no matter where it is stored in the storage hierarchy

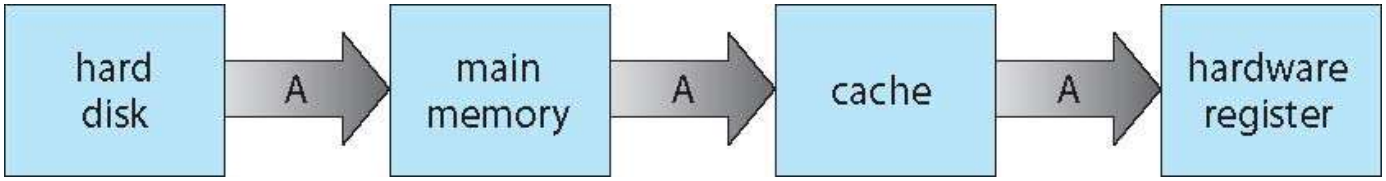


- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17

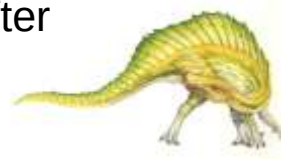


Migration of data “A” from Disk to Register

- Multitasking environments must **be careful** to use most recent value, no matter where it is stored in the storage hierarchy

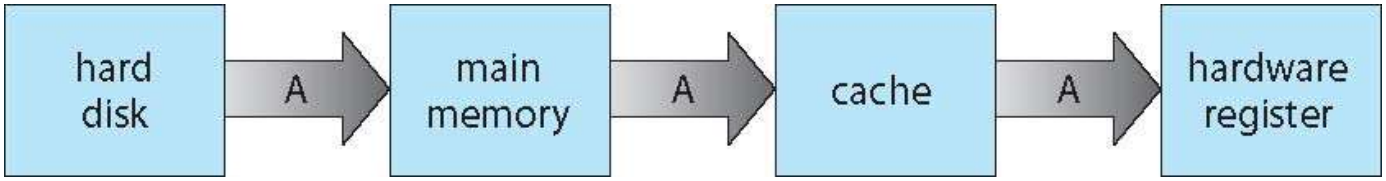


- For example, suppose that an integer A that is to be incremented by 1 is located in file B, B resides on disk;
- The increment operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory;
- This operation is followed by copying A to the cache and to an internal register
- Thus, the copy of A appears in several places: on the disk, in main memory, in the cache, and in a integer register
- On the increment takes place in the internal register, the value of A differs in the various storage systems; The values of A becomes the same only after the new value of A is written from the internal register back to the disk



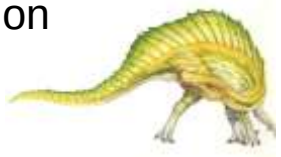
Migration of data “A” from Disk to Register

- Multitasking environments must **be careful** to use most recent value, no matter where it is stored in the storage hierarchy



- This situation becomes more complicated in a multiprocessor environment where, in addition to maintaining internal registers, each of the CPUs also contains a local cache.
- In such an environment, a copy of A many exist simultaneously in several caches.
- Since the various CPUs can all execute in parallel, we must make sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called **cache coherency**

- In a distributed environment, several copies of the same file can be kept on different computers



End of Chapter 1