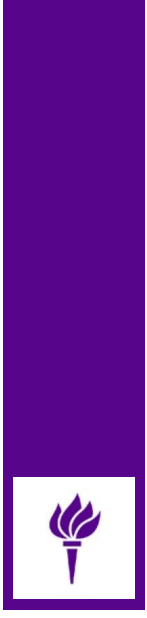


Operating Systems

Alex Delis
NYU-Abu Dhabi

January ’21

What is the job of an Operating System (OS)?

- A program that acts as an intermediary between a user of a computer and the hardware.
- Key Objectives:
  - Maintain control of computer resources.
  - Execute user-programs and make solving user-problems easier.
  - Make the computer system convenient to use.
  - Help author programs/utilities.
  - Utilize the hardware of the machine in an efficient manner.
  - Store both program and data.
  - Maintain use statistics and logs.

The Make Up of a Modern Computer System (CS)

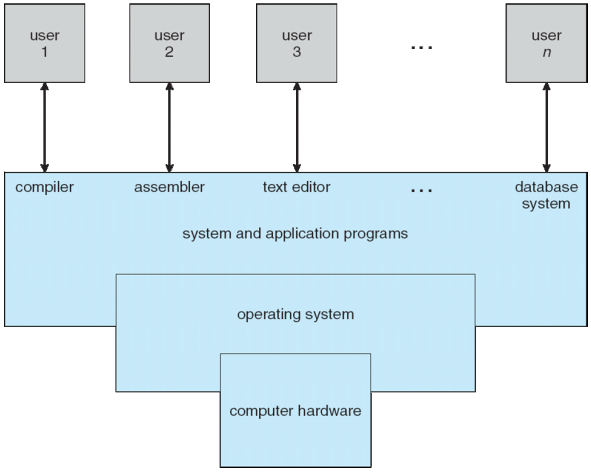
- A Computer System consists of 4 layers:
  - Hardware (physical devices, microprogramming and machine language):
    - CPU, Memory, Controllers, IO devices, NICs, etc.
    - Microprogramming: the fine-granularity software that interprets the low level instructions of Machine Language (ML).
    - Machine Language: repertoire of instructions that make the CPU to carry out specific tasks (i.e., *ADD*, *SUB*, *MOV*, *PUSHF*, *SHIFT*, *DEC*, *CMP*, *JMP*, *INT*, *CALL*, etc.)
  - Operating System (OS):
    - controls, shares and coordinates usage of hardware among users and applications.
    - hides complexity of lower levels (ML, Microprogramming and Hardware)

The Make Up of a Modern Computer System (CS)

■ And the other 2 layers are:

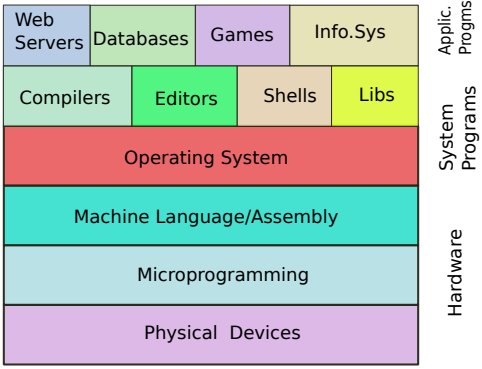
- Application and System Programs:
  - help solve specific jobs and computational problems
  - editors, compilers, assembles, shells, databases, word processors, browsers, games, information systems, web-servers, file manipulations utilities, device and network interface utilities, etc.
- Users:
  - individuals and other (networked) computer systems.

A Modern Computer System (CS)



The Layered Computer Systems (CS)

- A CS is constructed based on the principle of *abstraction*.



The Varying Nature of OSes

- In Different Circumstances, OSes Do Different Things:
  - For user-machines such as PCs and Workstations, convenience and ease of use are the predominant design goals for OSs.
  - Shared computers such as servers or mainframes must maintain satisfactory performance among all users.
  - Workstations have more sophisticated devices such as graphics cards and high-resolution screens in contrast to what happens with shared servers.
  - Handheld computer systems including modern phones, are optimized for good battery life.
  - Embedded systems found in a multitude of application areas such vehicles, airplanes, UAVs, telecomms, etc. require little or no user interface!

The OS Layer

The OS serves as both:

- a resource allocator that manages all requests issued by program; the latter do need CS computational/storage resources.
- a controller so as errors are avoided and if they occur, they are overcome in a gracious manner.

$\rightarrow$ In this context, OS *hides all complexities* of the underlying layers (i.e., physical devices, microprogramming, assembly language routines).

- The OS attains its "transparent" work through a dual mode of operation.
- The dual mode avoids users *tampering with* the internal of the OS and its underlying layers!
- An OS also functions as a border crossing or a wall that selectively let requests in.

The Dual Mode: Kernel vs. User Space

■ There is always a piece of code that runs on the computer and performs OS-related-work called the kernel.

- When the CPU runs kernel-instructions, the CS executes in kernel or supervisory mode.
- The kernel carries out chores on the behalf of executing programs that may involve both hardware and OS-software (modules).
- Everything else runs on user-mode and has to do with
  - system programs and utilities,
  - applications programs,
  - daemons, and libraries.

Kernel vs. User-Mode

- When a programs runs in “user-mode”, it cannot modify any of the structures/modules the kernel has control over.
- For instance when in user-mode, a program cannot write the interrupt-service handler for the system-disk – this is an unmodifiable part of the kernel!
- If one so desires can edit her program with `vim` and then switch to `emacs`
- The change of editors entirely occurs in user mode.

Going from User to Kernel Mode

- Assume the execution of a `C` program has advanced into the following point:

```
          ...
PC-->>    fread(buffer, strlen(c)+1, 1, fp);
          ...
```

  - This is sinlge-action that seeks to read from the disk of the CS to main memory a number of bytes using the `buffer` (as a temp structure) and the pointer `fp` to a file.
  - One approach would be the user (program) to carry all the needed actions for this I/O.
    - Too complicated if you want to look at his fine granularity (dealing with device, internal of disk, buffering, transfer, placement in main-memory etc).
    - Too much for the average programmer to deal with!

User- to Kernel-Space and Back!

- The OS offers this interface: `fread()`.
  - Once in this routine, the CPU is “taken away” from the program and is “given” into the kernel.
  - The kernel then executes the code that corresponds to the `fread()` on behalf of the user program.
  - This is the transition from user-mode $\Rightarrow$ kernel and the CPU runs code to complete the delegated work.
- Once OS kernel completes its work, the control is brought back to the user program by setting the CPU to execute the next instruction coming up in the program.
  - This is the transition from kernel $\Rightarrow$ user-mode and the CPU starts running the user program at the-next-to-PC instruction…

Things inevitably become messy!

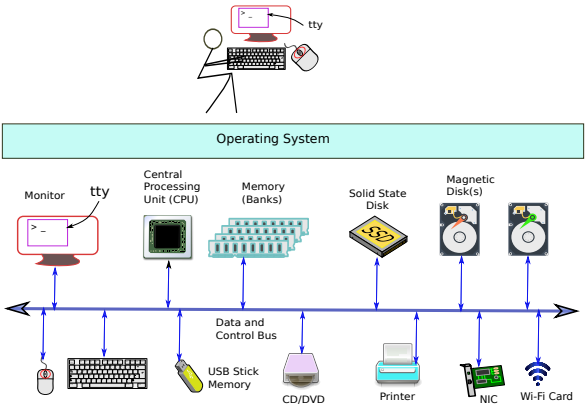
- As a CS may consist of multiple disks, multiple CPUs, memory banks, terminals (`ttys`), NICs, CD/DVD drives, printers, scanners, etc., the job of the OS is to provided and orderly and controlled allocation to the running programs!
- Problems that may easily arise:
  - Can many programs write at the same printer simultaneously?
  - How do we ensure that the main memory space used by a program is not *overwritten* by another?
  - How to push packets out of a NIC in an orderly manner and without (much) loss?

Starting Up an OS

- A boot-strap program is loaded any time a machine is powered up or reboot.
  - Typically stored in ROM or EPROM, generally known as firmware.
  - Initializes all aspects of system.
  - Loads operating system kernel and starts execution.
  - What is the difference between BIOS and this boot-loader program?

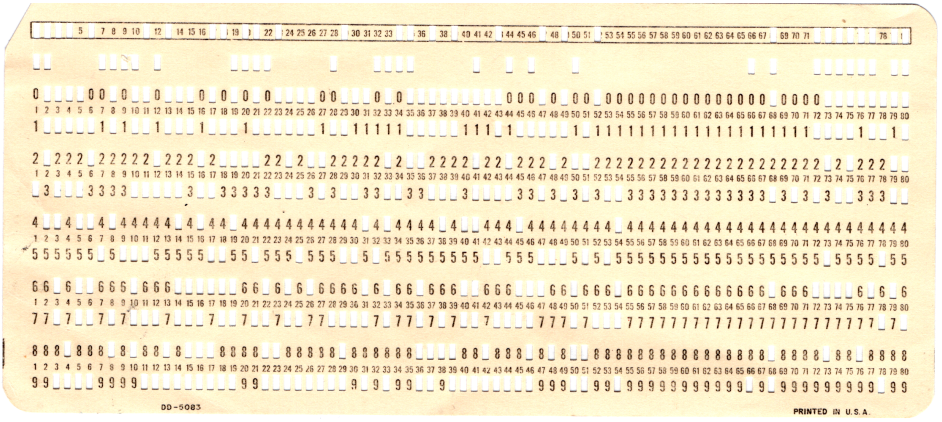
Computer System Organization

A CS is organized around a control and data bus.



Inputing a simple program statement… in 80s

- The mighty 80 characters instruction card!

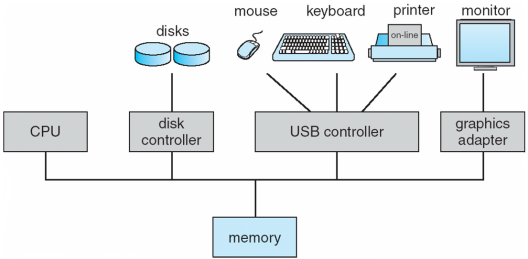


The early terminal (`tty`) that produced… punch cards!

- The mighty 80 characters instruction card!

CS Organization

- CPUs and dev-controllers connected via a bus yield access to shared memory.
- CPUs and Devs do compete for memory cycles!
- Controllers handle devices using the same IO interface such as SATA, SCSI, SCSI-wide, USB 2.0, etc.



- For this class, we assume that there is only 1 CPU!

CS Operation

- I/O devices and the CPU can function simultaneously.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- An I/O is mostly the time spent for data to “travel” from the device to local buffer of the controller.
- A device controller informs the CPU that it has finished its operation by causing an interrupt!

Interrupts

■ An interrupt is a hardware initiated signal that tries to get the attention to CPU so as the latter reacts to a *situation* that has just occurred.

- Through a interrupt vector, an interrupt transfers control of the CPU to a respective address that contains a piece of code that has to be executed in order to address the situation.
- This piece of code is called a service routine and it tells the CPU what to do when an interrupt happens.
- The code in question is also called *Interrupt Service Routine (ISR)* or *Interrupt Handler*.
- After the execution of *ISR*, control returns back to the main routine where work (execution) was interrupted.
- An interrupt architecture must save the address of the interrupted instruction.

Software Interrupts

■ A trap or an exception is a software-generated interrupt caused by either an error or a user request.

- Normal Interrupts: those caused by the software instructions; for example `IN/OUT` (or read/write statement).
- Exceptions: unplanned interrupts while executing a program; for instance, while executing a program if we get a division whose denominator is zero we reach an exception.

■ An Operating System is generally interrupt-driven.

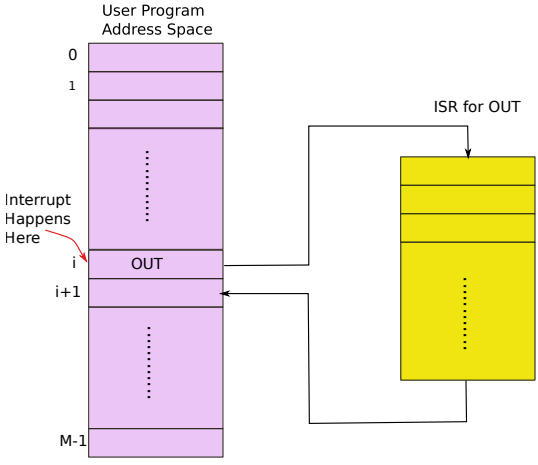
- The OS just waits and always reacts to signals/interrupts it receives!
- Once an interrupts is serviced, control returns to the user-program.

Interrupt Handling

- The OS has to preserve the state of the CPU by storing pertinent registers and the `PC`
  - The OS determines which type of interrupt has occurred through:
    - polling
    - vector-interrupt system.
  - A separate segment of code determines which action should be taken for each type of interrupt.

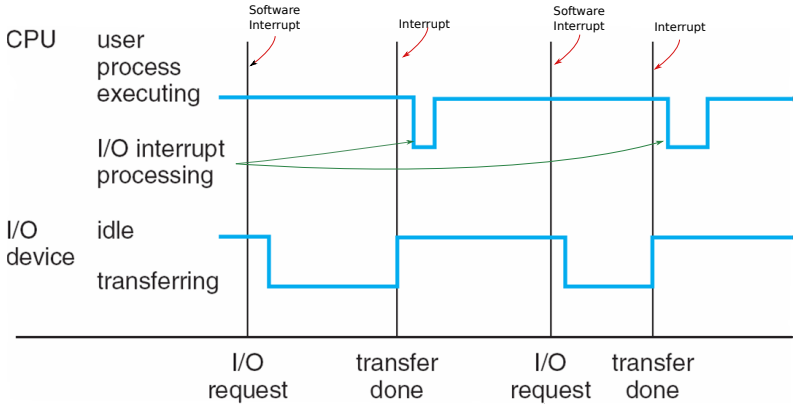
Management of CPU when an Interrupt Occurs

1. When an interrupt occurs the processor might be executing i^{th} instruction and the PC will be pointing to $(i+1)^{th}$ instruction.
2. When the interrupt occurs the PC value and registers are stored on the system stack.
3. The PC is now loaded with the address of ISR.
4. Once the ISR is completed the registers and address on the stack are pop out and placed accordingly.
5. Execution resumes at $(i+1)^{th}$ line of user program (function).



Understanding the Timeline of Interrupts

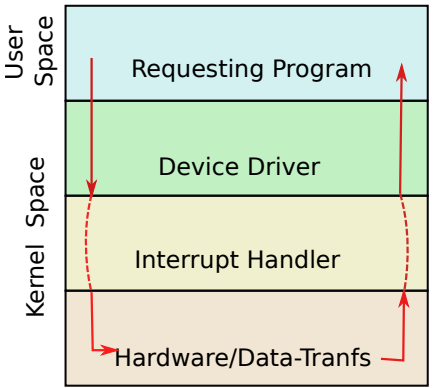
- Interrupts/Traps occur at unpredictable time instances and have to be treated.



Alternatives for Handling I/Os

- Device Drivers: routines that know the buffers, flags, registers, control bits and status information for a specific device works.
  - They are part of the kernel.
- After an I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- Synchronous I/O:

Synchronous I/O



■ Every word is transferred by the CPU with the help of the Device Driver until I/O completes.

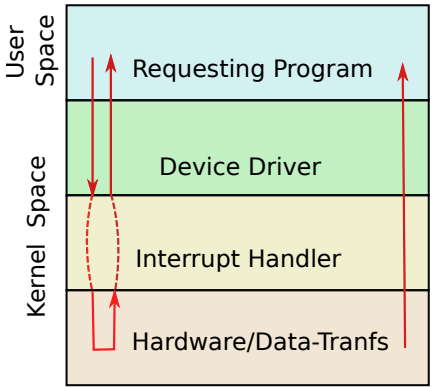
Alternatives for Handling I/Os

■ Once an I/O is launched, control *might return* to user program without waiting for I/O completion.

- This is also known as Asynchronous I/O.

- *System call*: request to the OS to allow user to wait for I/O completion
- *Device-status table*: contains entry for each I/O device indicating its type, address, and state
- OS looks up I/O *device-status table* to find out the device status and to modify table entry to include interrupt(s) that have likely ensued.

Asynchronous I/O



- The CPU notifies (sets up) the I/O.
- DMA carries out the transfer and issues an Interrupt for completion.

Definitions for Storage

- Bit: the basic unit of information - can be either 0 or 1.
  - A Byte is a set of 8 Bits arranged in sequence.
  - A Word is often a set of Bytes (either 4 or 8 Bytes).
  - Storage is mainly organized using Bytes – Words are mainly used for CPU ops.

Size	**Term**	Bytes
Kilobyte	KB	$1,024^1$ Bytes
Megabyte	MB	$1,024^2$ Bytes
Gigabyte	GB	$1,024^3$ Bytes
Terabyte	TB	$1,024^4$ Bytes
Petabyte	PB	$1,024^5$ Bytes
Exabyte	EB	$1,024^6$ Bytes
Zettabyte	ZB	$1,024^7$ Bytes
Yottabyte	YB	$1,024^8$ Bytes

Main Memory vs. Disk Storage Structures

- Main memory set up in banks can be accessed by the CPU directly.
  - It is a Random Access type of medium.
  - It is volatile.
- Hard disks: often metal or glass platters covered with magnetic recording material.
  - Non-volatile storage that serves as extension of main memory.
  - Disk surface is logically divided into tracks, which are subdivided into sectors.
  - The disk controller determines the logical interaction between the device and the computer.

Main Memory vs. Disk Storage Structures

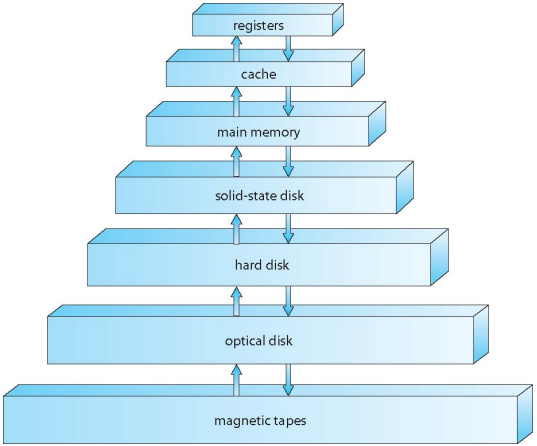
- Solid-State Disks or SSDs: storage media that are faster than hard disks (HDDs) and are non-volatile.
  - Flash memory serving as a disk.
  - Quick access time and lower latency than HDD
  - Smaller in size and more expensive than HDDs.
- Secondary Memory: all those devices that can hold data and are non-volatile.
  - HDDS and SSDs.
  - DVDs and CR-ROMs.
  - Tape-drives.

Storage Hierarchy

- Storage is routinely organized in hierarchy.
  - Parts of the hierarchy differ in
    - Speed - Size - Volatility - Cost
  - Caching also appears often in such hierarchies:
    - It copies data into faster elements of the hierarchy.
    - Main memory can be viewed as a caching mechanism for disk data.
    - Cached data are mostly transient pieces of information.
  - Device Driver: is provided for a each device to properly handle IO operations.
    - Offers uniform interface between controller and kernel of the OS.
    - “Talks” the connecting protocol of the device: SCSI, SATA, SAS, Ultra-SCSI, etc.

The Memory Hierarchy

- Registers are the *fastest* yet the *smallest* in-size medium for storage.



- Tapes the *slowest* yet can be *massive* in-size.

Caching

■ *Data-to-be-used* are copied from slower to faster storage temporarily and remain there for some time.

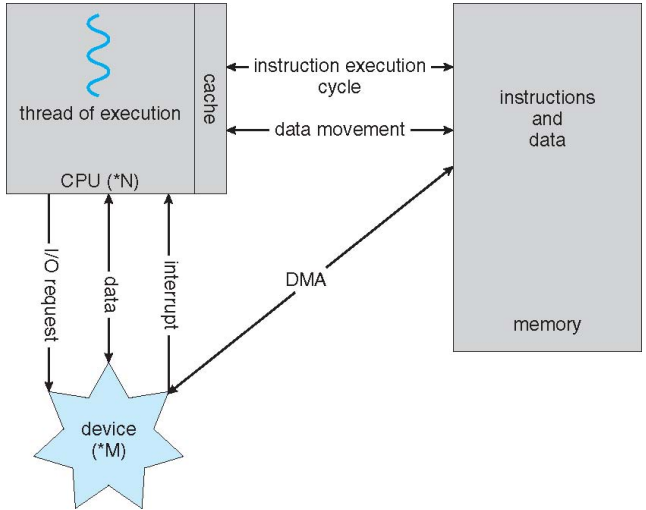
- Caching is an approach applicable at many levels of a CS: hardware, OS, software, and applications.
- Modus Operandi: cache is first checked to determine whether the sought piece of information is available there:
  - If it is, information is directly used from (faster) cache.
  - Otherwise, data is copies to cache from slower media and is then it is used (and remains in cache for some time).
- Cache is often inherently small and faster than the storage medium being cached.
  - Cache management becomes important design problem.
  - A replacement policy has to be adopted as necessitated by the limited size.

DMA: Direct Memory Access

■ A DMA-enable Device Controller directly transfers blocks of data from buffer storage to main memory without CPU intervention.

- It is used for *high-speed IO devices* able to transmit information at close to memory speeds including HDDs, SSDs, DVDs, etc.
- Key Principle: the controller generates only one interrupt per block, rather than the one interrupt per byte.
- The alternative would have been to have the CPU oversee the IO transfers on a Byte by Byte (or Word by Word) fashion.
  - Significantly slower for checks have to happens numerous times.
  - More importantly, CPU is busy and cannot do anything else while the IO occurs.

DMA Operational Diagram



CS Architectures

■ In their majority, CS feature 1 general-purpose processor.

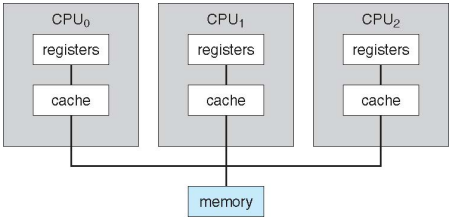
- Often, they have special-purpose processors as well.

■ Multiprocessors systems are both important and popular; a.k.a., parallel or tightly-coupled systems.

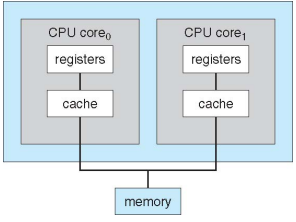
- Increased throughput and shorter task response times
- Economy of scale and increased reliability (including grateful degradation or fault-tolerance).
- 2 types:
  - Asymmetric Multiprocessing: each CPU is assigned a specialized task.
  - Symmetric Multiprocessing: all CPUs can execute all tasks.

CS Architectures

- Symmetric Multiprocessing
  - Multiple CPU Boards have access to the same memory.

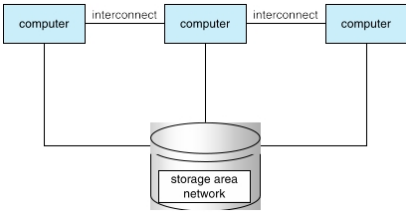


- Multi-core Architecture
  - Chassis containing multiple separate systems or core.



Clustered Systems

- Clustered Systems: multiple networked computer systems working together.
  - They share storage via a SAN (Storage Attached Network).
  - Offer highly-available services surviving failures:
    - Asymmetric Clustering: 1 machine is on stand-by.
    - Symmetric Clustering: machines run apps and monitor each other.
  - Such clusters are mostly for HPC and use parallelization.

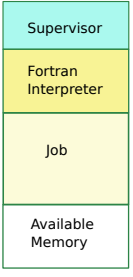


OS Innovations

Batch Systems

- In early OSs, 1 job at a time was provided to the machine mostly in manual fashion.
  - Most of the tasks were programs written in `Fortran`, `Cobol` or `Assembly`.
  - Each different type required its own interpreter.
  - Interpreters of compilers were loaded onto the machine using tape-reals.

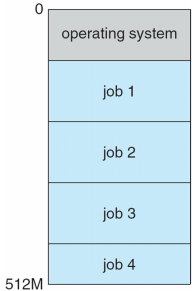
- Batching had similar tasks (say for just `Fortran` programs) pulled together and have them automatically fed into the CS one at a time.
- This was carried out with the extension of the supervisor to include device manipulation instructions by the operators.
  - `JCL`: job control language (an early `shell`)
- Batching led to shorter times for jobs to produce their output.



Single-task Memory Model

OS Innovations

- Multiprogrammed OSs.
  - CS that work with exclusively 1-user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so as CPU always has 1-task to execute.
    - A number of all jobs ready for execution are kept in memory at the same time.
    - 1-job gets selected and run via scheduling
    - When a job has to wait, say for an IO, the kernel switches to another job available job in main-memory.



Memory Model for Multiprogramming

OS Innovations

■ Timesharing or Multitasking: the CPU does not stay indefinitely with 1 job.

- If needed, CPU is taken-away (pre-emptied) from the currently executing job once the latter has progressed for a period of time (*time-slice*) so as fairness among all jobs can be attained. ⇒ the CPU is given to another awaiting job.
- Interactive Computing: perception that user has the entire machine to herself.
- Doing this trick, the response time can be less than 1 second.
- If multiple jobs are available, who take on the CPU for the next time-slice is determined by the scheduler.

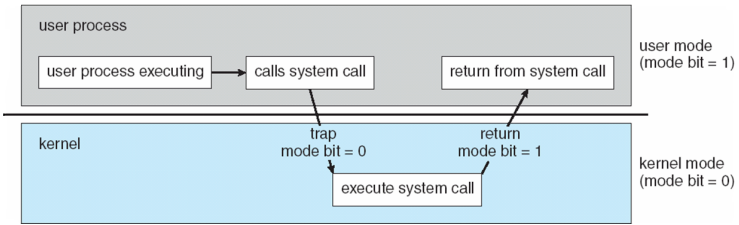
■ Swapping allows for a task that does not temporarily fit in main-memory (MM) to move out to the HDD.

■ Virtual Memory allows for the execution of jobs that do not fit in their entirety in MM.

The Dual-Mode of Operation

■ As the CPU has to travel between user-programs and kernel space in memory, the dual-mode is used as the principle feature to offer OS protection.

- 2 modes: user-mode and kernel-mode.
- The *mode-bit* courtesy hardware, offers:
  - capability to *distinguish* when system run user code or kernel code.
  - Some instructions are *privileged* and can be executed only in kernel mode.
  - System calls often change the mode from user-to-kernel and upon return the mode is reset to user (again).



Timers

- A timer prevents infinite loops from occurring or processes to hog resources.
  - Timer is set to interrupt the CPU after a specific time period.
  - Keep a counter that is decremented by the physical clock.
  - The OS can set a counter; this is a privileged instruction.
  - When counter becomes zero, an interrupt is generated.
  - Set a counter up before scheduling process to regain control or terminate program that exceeds allotted time.

Work for Kernel: Process Management

- A process is a logical unit work in OS and nominally, a program in execution.
  - Source program is a *passive entity* while process is an *active entity*.
  - A process calls for resources to do her work: CPU, MM, HDD, files, printer, etc.
  - When a process exits, the kernel has to reclaim resources.
  - Single-threaded process has one program counter (`PC`) specifying location of next instruction to execute.
  - Process executes instructions sequentially, one at a time, until completion.
  - An OS has many processes, some user, some kernel.
  - When multiplexing a CPU (or multiple) with processes, we have what is termed concurrency.
- Multi-threaded Processes
  - There is a `PC` for every thread.

Work of Kernel: Process Management

- The kernel/OS is responsible for all the following management activities:
  - Create and purge users/system processes.
  - Suspend and resume processes when needed/asked.
  - Provide synchronization mechanisms for 2 or more processes.
  - Offer mechanisms for processes to communicate with others.
  - Provide mechanisms for managing deadlocks among processes.

Work of Kernel: Memory Management

- For a program to be executed, its entire (or part of code) has to be in MM.
  - MM Management determines what, when and for how long is in MM.
    - This is done so that use of CPU and MM are optimized.
  - Activities that have to do with Memory Management:
    - Keeping track of which parts of memory are currently being used and by which process.
    - Determining which processes (or parts thereof) and data to move into and out of memory.
    - Allocating/Deallocating memory space to processes when required.

Work of Kernel: Storage Management

- OS provides uniform, logical view all information/data storage.
  - Through the notion of file abstracts physical elements to a logical storage unit.
  - Each medium is controlled by a pertinent device: HDD, SSD, tape, etc.
  - All the above media have varying access speed, capacity, latency, data-transfer rates, access methods on-the-device (sequential or random).
- The management of files is undertaken the File-System (FS):
  - Files can be organized in directories.
  - Directories may be nested within other directories created a tree-like acyclic logical layout on the medium.

Work of Kernel: Storage Management

- Kernel operations that are applicable to FS entail:
  - Creating and deleting files and directories.
  - Primitives to manipulate files and directories (i.e., move around in the logical structure of the FS).
  - Mapping files onto secondary storage.
  - Providing security for elements of FS belonging to different users.
  - Backing-up files and dir structures onto stable (non-volatile) storage media.

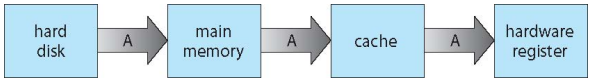
Work of Kernel: Key Characteristics of Various Storage Levels

Key Features of Various Storage Options

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

Data Consistency Issue Among the Layers

■ In multiprogrammed/multitasking environments, we must be aware that the value of an object `A` might be different at various layers of memory.



- Multiprocessor CSs must provide for cache consistency in hardware so that all CPUs "see" the same value.
- In distributed systems, attaining consistency is a more difficult problem: several copies coexist and there must be a way to decide which one is the correct!

Work of Kernel: Mass-Storage Management

■ HDDs often store data that a) do not fit entirely in MM or b) must be maintained for protracted periods of time. ⇒ Proper Management of Data-Space becomes critical.

- Speed of the entire CS hinges on how the HDDs/SSDs/etc. and respective algorithms behave.
- Pertinent Kernel-activities entail:
  - Free-space management,
  - Storage allocation
  - Disk-scheduling.

■ Some storage does not need to be fast but massive: tapes a.k.a. tertiary memory.

- Tertiary Memory (TM): Optical storage and magnetic tapes.
- Optical can be WORM/RW: write-once-read-many, read-write and variations.
- TM has to be managed by OS and/or applications.

Work of Kernel: IO Subsystem

■ In its effort to hide peculiarities, the OS's IO-subsystem undertakes the following responsibilities:

- Main-Memory (MM) management.
- Buffering: storing data temporarily in MM or while in transfer.
- Caching: storing parts of data is faster devices.
- Spooling: addressing the overlapping requests for printing job outputs.
- Management of dev-drivers for hardware-devices and their Interfaces.

Work of Kernel: Protection and Security

■ Protection Mechanisms are deployed to control access by processes to both physical and logical OS resources.
OS generally distinguish among user classes, to determine who can do what:

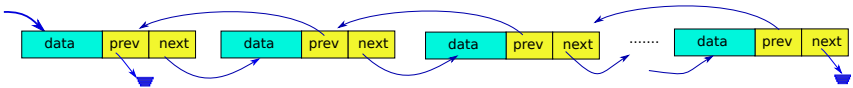
- Users are issued authoritative identities (ie, user-ids/security-ids).
- A specific user-id is associated with all files & processes of that user to determine access control.
- Group-IDs allow sets of users to be defined. A group-ID permits processes and files to be accessed from the respective set of users.
- Privilege escalation allows a user to change to “effective-ID” that may provide additional rights.

■ Security is the collective set of constructs that safeguard an OS from external and internal attacks.

- Examples: denial-of-service, worms, viruses, identity theft, theft of service, etc.

Kernel: Data Structures used to Accomplish the Work

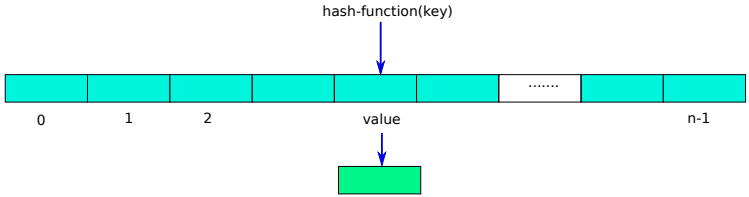
- An assortment of standard programming data structures you already know:
  - Arrays.
  - Single-linked lists.
  - Circular-linked lists.
  - Double-linked lists.



  - Binary Search Tree: left-val $\leq$ right-val – $\mathcal{O}(\log n)$

Kernel: Data Structures used to Accomplish the Work

- Hashing function can create a hash-map



- Bitmap: string of n binary digits representing the status of n items (on or off).
- `/usr/src/linux-headers-5.4.0-26/include/linux/list.h`
- `/usr/src/linux-headers-5.4.0-26/include/linux/kfifo.h`

Traditional and Not-So-Traditional CS Environments

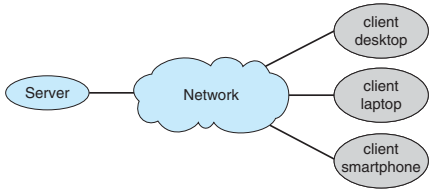
■ OSs were initially developed for stand-along systems but they have significantly expanded their realm of operations:

- Web Portals and Servers (Apache2, WebSphere, Ngix, etc.)
- Thin-clients (Chrome).
- Laptops and handheld mobile pads (iOS-devices, Android, etc)
- All the above networked via wired or wireless LANs/Cellular-Networks.
- Networking offers data-comm paths often through `TCP/IP` protocol.

■ Distributed Computing Systems: Collection of separate, possibly heterogeneous, systems networked together.

Traditional and Not-So-Traditional CS Environments

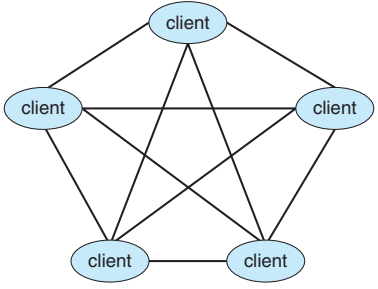
■ Client-Server Computing: a server offers a specialized service; for instance, the server run a `PostgreSQL` database engine.



- The server responds to requests made by clients through a data-network.
- Clients can be desktops, laptops, phones, or even terminals.
- The rendered service could be: file-service, web-service, database, information-server, etc.

Traditional and Not-So-Traditional CS Environments

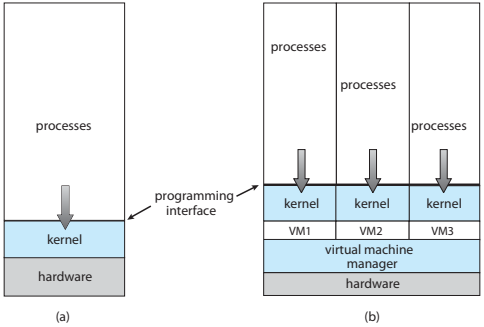
- P2P Systems do not distinguish among clients and servers.
  - All nodes are peers serving as clients, servers or both.
  - A node must registers its service with a central look-up authority.
  - A node broadcasts its services and requests services through a discovery protocol.



Overlay-network of Nodes.

Traditional and Not-So-Traditional CS Environments

- Virtualization allows OSs to run applications within other OSes.
  - Emulation used when source CPU is different from target (Arm to x86).
    - When language not compiled in native code it is interpreted $Rightarrow$ Slower.
  - Guest OSes natively compiled are running natively on host OSes.
  - VMM (Virtual Machine Manager) offers virtualization services.



Traditional and Not-So-Traditional CS Environments

- Real-time OSes empower a wide range of embedded systems in broad consumer use.
  - Requests to OS have to comply with strict timing constraints.
  - An operation is deemed correct only if it is met within the constraint.
- Cloud Computing offers computing, storage, even apps as a service across the network.
  - Cloud Computing extensively uses virtualization.
  - Services rendered:
    - Software as a Service (SaaS): a word processing facility.
    - Platform as a Service (PaaS): a database or an information service.
    - Infrastructure as a Service (IaaS): compute and storage nodes.

Alex Delis, alex.delis -AT+ nyu.edu

NYU Abu Dhabi