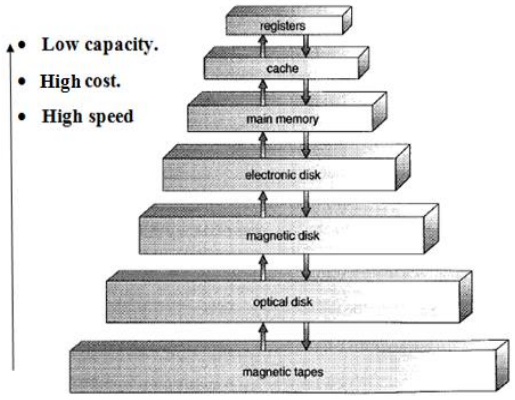
1. The major operating systems components:
   - Kernel: is responsible for low-level tasks such as memory management, CPU scheduling, device I/O, and handling systems calls.
   - File System management: is responsible for organizing and managing the storage of files and directories on disk or other storage devices.
   - Process Management: is responsible for creating, scheduling, and managing processes, which are instances of running programs.
   - Memory Management: is responsible for handling the allocation and deallocation of memory resources for processes, and manages the mapping between virtual and physical memory addresses.
2. Describe the basic computer system organization:
   - The CPU processes instructions and performs calculations.
   - The memory stores data temporarily(RAM) and permanently(ROM).
   - The input devices allow users to input data into the computer, and the output devices present processed information to users.
   - Storage devices store data and programs permanently.
   - The system bus connects CPU to other components, allowing data transfer.
   - The motherboard is the main circuit board that connects all components.
3. Operating Systems: acts as an intermediary between a user and the computer hardware

3.2. Essential functions/services provided by operating systems for systems:
   - Resource management
   - Process management
   - Memory management
4. Operating systems 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
5. A computer system can be divided into four components:
   - Hardware: provides basic computing resources e.g CPU, Memory, I/O devices
   - Operating system: Controls and coordinates the use of hardware among various applications and users
   - Application program: defines the ways in which the system resources are used to solve the computing problems of the users.
   - User: Interacts with the computer system through input and output devices.
6. OS as a resource allocator:
   - Manages and allocates hardware resources among competing processes and users
   - Decides between conflicting requests for efficient and fair resource use
7. OS as a control program:
   - Regulates and supervises the execution of other programs and operations within the computer system.
8. NB: It is the one program running at all times on the computer.

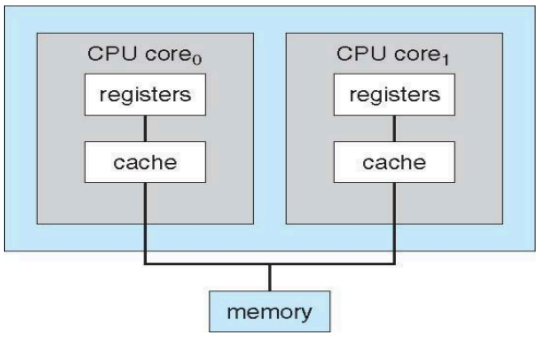
9. Computer Startup: bootstrap is loaded at power-up or reboot. It is typically stored in ROM or EPROM, generally known as the firmware. It initializes all aspects of systems, loads the operating system kernel, and starts execution.
10. I/O devices and the CPU can execute concurrently:
    - This concurrent execution is facilitated by the presence of device controllers, each responsible for managing a specific type of I/O device.
11. I/O Operations: the device driver loads appropriate registers within the device controller, and data is transferred from the device to the local buffer of the controller. Then the device controller informs the CPU that it has finished its operations by causing an interrupt.
12. Interrupt: is a signal sent by a hardware or software to the CPU, requesting its attention and temporarily halting its current execution to handle a specific event or condition.
12.1. Interrupts are used to facilitate the handling of asynchronous events such as:
    - Completion of an I/O operation - a hardware error - timer expiration
13. Common functions of interrupts:
    - Handling I/O operations
    - Timer and clock management
    - Error and exception handling
14. Lost interrupt: occurs when an interrupt signal fails to be recognized or serviced by the CPU.
15. How lost interrupt is prevented:
    - Implement error checking codes
    - Use redundant signaling paths
    - Prioritize critical interrupt signals
    - Employ hardware handshaking protocols
16. A trap: is a software-generated interrupt caused either by an error or a user request.
17. Meaning of "the OS is interrupt driven": it means that the OS relies heavily on interrupts to manage and respond to events and activities within the computer system.
18. Interrupt Handling: The OS preserves the state of the CPU by storing registers and the program counter. It determines which type of interrupt has occurred whether polling or vectored interrupt system. Separate segments of code determine what action should be taken for each type of interrupt.
19. Types of I/O Structure:
    - Synchronous:
      - in synchronous I/O, control returns the user program only after the I/O operation has completed
      - A "wait" instruction is used , which essentially idles the CPU until the I/O operation is completed.
    - Asynchronous:
      - in asynchronous I/O, control returns to the user program immediately after initiating the I/O operation

- The program can continue executing other tasks while I/O operation proceeds in the background.

20. Direct Memory Access (DMA): is a method used by high-speed I/O devices to transfer data efficiently.
    - The device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
21. The only large storage media that the CPU can access directly is the: Main Memory
22. The extension of the main memory that provides large non-volatile storage capacity: Secondary Memory
23. Description of magnetic disks: it is a rigid metal or glass platter covered with magnetic recording material.
    - The disk surface is logically divided into tracks, which are sub-divided into sectors
    - The disk controller determines the logical interaction between the device and the computer.
24. Storage system are organized in hierarchy based on the following:
    - Speed - Cost - Volatility
25. Caching: refers to the technique of storing frequently accessed data in a temporary location with faster access speeds.
26. Diagram of storage-device hierarchy:



27. Multiprocessors, also known as parallel systems, or tightly-coupled systems: is a computer system that contains multiple processors that are tightly interconnected and capable of executing tasks concurrently.
28. Advantages of multiprocessors:
    - Increased throughput
    - Increased reliability- fault tolerance
    - Economy of scale
29. Types of multiprocessing:
    - Asymmetric Multiprocessing:
    - Symmetric Multiprocessing:
30. A dual-core design:



31. Clustered Systems: it is a system where multiple independent computer systems called nodes, are interconnected to work together as a single unified system.
32. Clustered systems usually share storage via a: storage-area network(SAN)
33. Advantage of clustered systems:
    - High availability
    - Improved performance
    - Simplified management
34. Asymmetric and Symmetric clustering: asymmetric clustering has one machine in hot-standby mode, while symmetric clustering has multiple nodes running applications and monitoring each other.
35. Multiprogramming: is a computer OS system technique where multiple programs are executed concurrently on a single processor.
36. Advantages of multiprogramming:
    - Increased CPU utilization
    - Better responsiveness
    - Resource sharing
37. Dual mode operation: refers to the CPU's ability to switch between two privilege levels; the user mode and the kernel mode.
    - ensures stability - ensures security - ensures efficient resource management
38. User mode: in this mode, applications run with limited access to system resources.
    - This limits program actions and prevents system crashes or interference.
39. Kernel mode: the OS itself operates in kernel mode. In this mode, it has unlimited access to system resources.
    - It allows the OS to manage processes, allocate memory, and handle system calls from user applications.
40. Mode-bit: indicates whether the system is currently operating in user mode or kernel mode.
41. Some instructions designated as privileged, are only executable in kernel mode.
42. Why the transition from User to Kernel Mode happens:
    - System calls - Interrupts
43. Process hogging resources: refers to a program or task that consumes an unusually large amount of a computer's resources.

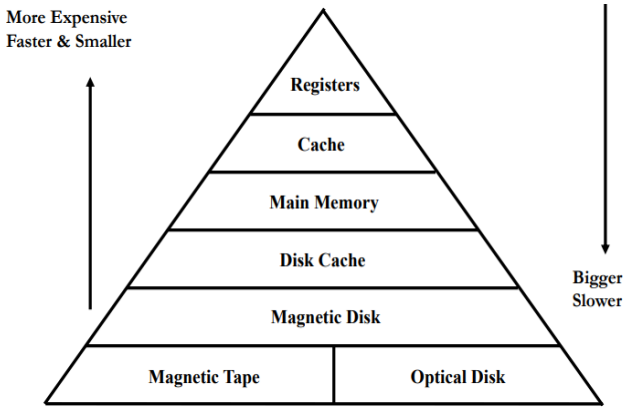
44. Instruction execution: instructions are stored in memory and retrieved by the CPU one by one with the following registers.

- The program counter: keeps track of the memory address of the next instruction to be fetched.
- Instruction register: Once the CPU fetches an instruction from memory using the PC, it stores that instruction in the IR. The IR temporarily holds the instruction while it's being decoded.
- Memory address register: holds the memory address from where the instruction needs to be fetched or stored. During instruction execution, the PC value is copied to the MAR to fetch the instruction from memory.
- Memory buffer register: holds the instruction that is fetched from memory or is waiting to be written back to memory. After the CPU fetches an instruction using the MAR, it places the instruction in the MBR.
- I/O address register: holds the address of an I/O device with which the CPU needs to communicate.
- I/O buffer register: holds the data that needs to be transferred between the CPU and an I/O device.

47. Registers that deal with memory interactions during the instruction executions:

- Program Counter (PC) - Instruction Register (IR) - Memory Address Register (MAR) - Memory Buffer Register (MBR)

48. Diagram of a memory hierarchy:

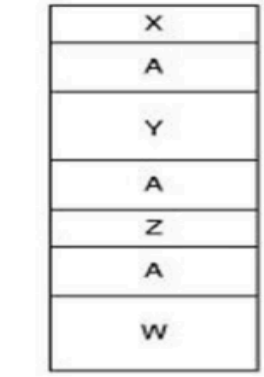


49. Locality: refers to the tendency of a processor to access data and instructions in a non-random way.

50. The two main types of locality:

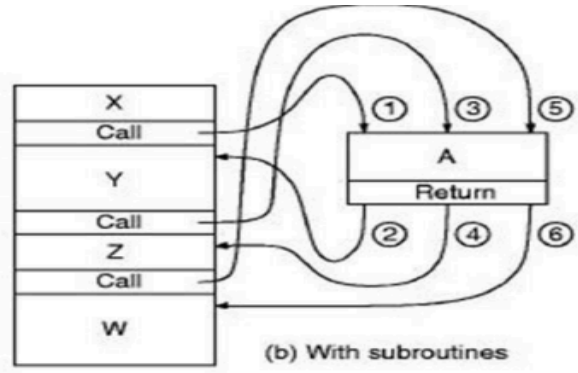
- Temporal locality: refers to the tendency to access the same data or instructions repeatedly within a short period. For example, when a loop is running, the CPU will access the same set of instructions over and over again.
- Spatial locality: refers to the tendency to access data or instructions that are located close together in memory. For example, when a program reads an element from an array, it's likely to access the next element in the array soon after.

51. Does the OS directly create locality?: The operating system doesn't directly create locality, but it can leverage the principle to improve overall system performance.
52. The call/return mechanism or procedure/function call: refers to the process of invoking a block of a code (a procedure or function) during program execution and then returning control back to the point in the program from which the call was made once the procedure or function has completed its task.
53. Diagram of call/return mechanism:



(a) Without subroutines

Faster programs.
Less overhead.



(b) With subroutines

Smaller programs.
Easier to maintain.
Reduces development costs.
Increased reliability.
Fewer bugs do to copying code.
More library friendly.

54. The principal requirements for memory management:
- Relocation: allows the OS to load a program at any free memory location and adjust its memory addresses at runtime. This ensures the program runs correctly regardless of its physical location.
- Protection: prevents the risk of one program overstepping its bounds and modifying another program's memory space by defining access rights for different memory regions.
- Sharing: allows controlled access to specific memory regions by authorized processes, promoting resource utilization.

55. Memory partitioning: is a memory management technique in OS used to manage how programs access the computer's main memory (RAM).
56. Reason for memory partitioning: Without partitioning, only one program could reside in memory at a time. Partitioning allows the operating system to load multiple programs into different regions of memory, enabling them to share the CPU and run concurrently.
57. There are two main memory partitioning techniques:
- Fixed/static partitioning: the memory is divided into fixed-size partitions during system startup, and each partition can hold one process.
  - Advantages: simple to implement, fast memory allocation
  - Disadvantage: limited flexibility, internal fragmentation

- Dynamic partitioning: the memory is divided into variable-sized partitions during program execution to accommodate processes of varying memory requirements.
  - Advantages: Eliminates internal fragmentation, no restriction on process size
  - Disadvantages: Complex memory allocation, external fragmentation

58. Memory management techniques:
    - Memory allocation strategies - memory partitioning - memory paging - memory segmentation

59. Paging: is a memory management technique that divides both physical memory (RAM) and logical memory (program) into fixed-size blocks called frames and pages, respectively.

60. How paging works:
    - RAM is divided into equal-sized frames, and processes are divided into equal-sized pages.
    - A page table keeps track of which page is in which frame (or not in memory at all)
    - When a process tries to access memory, the memory management unit (MMU) uses the virtual address to look up the corresponding page number in the page table.
    - The MMU then uses the page number to find the frame number in the page table, which tells it where the data is located in physical memory (or flags a page fault if the page isn't loaded).

61. Advantages of paging:
    - Better memory allocation
    - Faster address translation
    - Support for virtual memory

62. Segmentation: is a memory management technique that divides a process's memory into logical chunks called segments.

63. Difference between paging and segmentation: paging uses fixed-sized units while segments can have variable sizes based on the program's structure.

64. How segmentation works:
    - A program is divided into logical segments based on its functionality (code, data, etc)
    - A table keeps track of information about each segment, including its base address (starting location in memory) and size limit.
    - When the CPU generates an address, it's a logical address with a segment. The segment table is used to translate this to a physical address in memory. This translation involves adding the base address of the segment to the offset within the segment specified by the logical address.

65. Advantages of segmentation:
    - Better memory utilization
    - Modular protection
    - Logical organization

66. key security issues and remedies related to memory management.
    - Buffer overflows:

Remedy: Use bounds checking to ensure data fits within the buffer

- Memory leaks

  Remedy: Implement memory profiling to identify and fix leaks.

- Memory corruption:

  Remedy: Utilize memory error detection tools like AddressSanitizer.

67. Linking and loading: are two crucial behind-the-scenes processes that work together to prepare the program for execution by the OS.
68. Linking: is the process of combining multiple object files and libraries to create a single executable file or shared library.
69. Loading: is the process of placing the linked program into the computer's main memory where it can be accessed by the CPU for execution.
70. Virtual-memory paging: paging, but not all pages need to be in memory at one time.
71. Virtual memory segmentation: segmentation, but not all segments need to be in memory at one time.
72. Buddy system: involves dividing memory into fixed-size blocks and then allocating and deallocating these blocks in a way that minimizes fragmentation.
73. How the buddy system works:
    - The system divides the memory into fixed-size blocks, often powers of 2.
    - When a process requests memory, the buddy system searches for the smallest available block that's large enough to hold it.
    - If the requested memory size perfectly matches a block size, that block is allocated to the process.
    - If a larger block is needed, the buddy system splits the block in half until it finds a block that fits the request.
    - The leftover block (the buddy) is marked as free for future allocation.
74. Characteristics of an executing program:
    - Instructions must be in main memory
    - Uses CPU time, memory, and other resources to get its job done.
    - Instructions are followed one after another, unless there's a jump or loop.
75. What problems result from lack of memory:
    - Program crashes
    - Slow performance
    - Inability to Handle Large Data Sets
76. What if the program needs to grow while executing?: Many programming languages such as C/C++ and Java support **dynamic memory allocation**, allowing programs to request memory during runtime. This is useful for programs whose memory requirements are not known beforehand.
77. How about moving to a new machine?: Programs can be serialized into a byte stream containing their state (memory and variables). This serialized data can be transferred and

deserialized on the new machine to resume execution.This process is commonly used for saving program state or transferring data between machines.

78. Advantages of NOT requiring all of an executing program to be in physical memory:
    - Larger program address space.
    - More executing program memory.
    - Less I/O needed to get a process going.

79. Real memory: is the physical memory that provides fast access for the CPU to store and retrieve data currently being used.

78. Virtual memory: is a memory management technique employed by the OS to create the illusion of having more memory than physically available.

80. What is the difference between “real” and “virtual” memory?:

Real Memory	Virtual Memory
Physical memory	Memory management technique
faster	slower
Limited by installed RAM	Limited by storage space

81. Cache memory: provides illusion of very high speed.

82. Principle that makes Virtual Memory possible?: Principle of locality - programs frequently reference data and instruction that are nearby in memory.

83. Part(s) of a program that needs to be in memory?: Resident set - actively used portions of a program that reside in real memory at any given time.

84. Problems that might occur with virtual memory?:
    - Thrashing- when the OS is constantly swapping pages between RAM and disk due to insufficient real memory.
    - Fragmentation - overtime, as the OS swaps pages in and out of memory, free memory can become fragmented.

85. Memory Management Unit(MMU): is a hardware that acts as a translator between the virtual memory addresses used by programs and the physical memory address of the actual RAM.

86. How MMU works:
    - The MMU intercepts memory access requests from the CPU. It uses a translation table stored in memory to convert the virtual address into a physical address in RAM.
    - Once the MMU has the physical address, it forwards the request to the RAM to access the desired data or instruction.

87. Page tables and where they are stored: they act as a **mapping** between virtual addresses used by programs and physical addresses in real memory (RAM).
    - Most frequently used page tables (active programs) reside in RAM for speed.
    - Less frequently used page tables are stored on disk (paging file) to save RAM space.

88. How is a program started:

- Start process with no pages in memory.
- Page fault occurs on memory access.
- OS identifies and locates missing pages.
- OS loads missing pages into memory.
- Memory table updates and program resumes.

89. At what point in a program's execution can a page fault occur?:
    - Swapping inactive data to disk to free up RAM
    - Dynamic memory allocation during program run-time
    - Trying to access memory outside allocated space

90. The worst case of page faulting: occurs when a program's memory access pattern leads to a high rate of page faults, severely impacting performance. May arise from the following factors:
    - Bad page replacement - Poor program locality - Limited RAM

91. Paging time:
    - Disk latency 8 milliseconds
    - Disk seek 15 milliseconds
    - Disk transfer time 1 milliseconds
    - Total paging time ~25 milliseconds

92. Formula to find effective access time:
    - EAT = (1-p) × ma + p × pft

      Where p is probability of page fault, ma is memory access time, pft is page fault time

Example:

EAT with 100 ns memory access and 25 ms page fault time:

$$EAT = (1 - p) \times ma + p \times pft$$
$$= (1 - p) \times 100 + p \times 25{,}000{,}000$$
$$= 100 + 24{,}999{,}900 \times p$$

If we fault 1 out of 1000 ($p = 0.001$) accesses, what is the EAT?

$$EAT = 100 + 24{,}999{,}990 \times 0.001$$
$$= 25\ \mu s \text{ (250 times slowdown!)}$$

How do we get less than 10% slowdown?

$$EAT \leq 1.10 \times 100\ ns = 110\ ns$$
$$100 + 24{,}999{,}990 \times p \leq 110\ ns$$

Less than 1 out of 2,500,000 accesses

93. Conversion cheats:
    - 1KB = $2^{10} bytes$, 1MB = 2^{20}bytes, 1GB = 2^{30}bytes,
    - 2^n values → n bits, n bits → 2^n values
    - Converting xGB to bytes: first convert x to base 2, then multiply by 2^{30}

94. Given a 4 Gigabyte (GB) address space with 4 Kilobyte (KB) pages and an 8-byte page table entry size, how much memory is required for the page table?

- *Total size = number of pages/entries × page size*
- Number of pages/entries:
    - *Address space size (bytes) / Page size (bytes)* = 2^32 bytes / 2^12 bytes = 2^22 entries
- Total size:
    - *Number of entries * Entry size (bytes)* = 2^22 entries * 8 bytes/entry = 2^25 bytes

96. Thread: is a basic unit of execution within a process. They are sometimes called lightweight processes because they share similar traits to processes but demand fewer resources for creation and management.

97. Benefits of multithreaded programming:

- Better user interaction
- Shared memory and resources
- Concurrent execution

98. Thread Cancellation: refers to the process of terminating a thread before it has completed its execution.

- Canceling thread: is the thread that initiates the cancellation request.
- Target thread: is the specific thread that is intended to be terminated.

99. Ways in which a thread can be canceled:

- Asynchronous cancellation: Immediate termination signal.
- Cooperative cancellation: Periodic termination checks.
- Voluntary exit: self termination.

100. Deadlock: refers to a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource held by some other process.

101. Conditions under which a deadlock situation may arise:

- Mutual exclusion - Hold and Wait - No Preemption - Circular wait

102. Methods for handling deadlocks:

- Ensure at least one of the Coffman conditions cannot hold
- Use Banker's algorithm to prevent a circular wait condition
- Allow deadlocks to occur, but have a mechanism to detect and recover from them.

103. Starvation: refers to a situation where a process is perpetually denied the resources it needs to proceed, because the resources are continuously allocated to other processes.

104. Operating system services for user:

- User interface
- File system management
- Program execution

105. Why a personal computer needs an operating system: it needs an OS to manage hardware and software resources efficiently, handle tasks like memory management and file storage, and

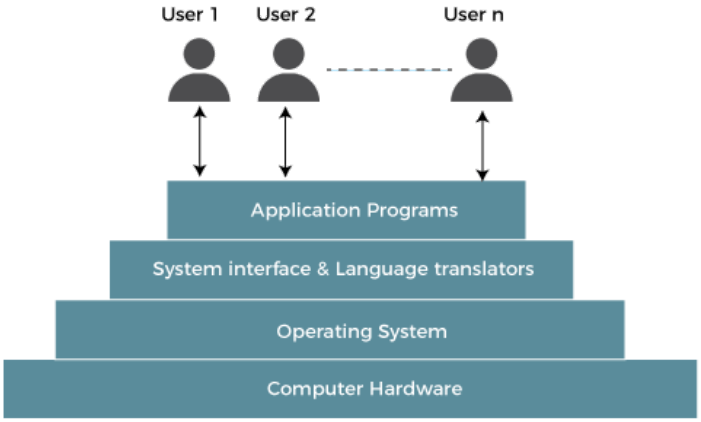
provide a user-friendly interface. Without it, interacting directly with hardware would be complex and impractical.

106. Difference between the system programs and application programs: system programs manage the computer's resources and hardware while application programs are the ones you directly use to perform specific tasks.

107. Why it is not possible to make programs and data to reside in main memory permanently:

- Main memory is volatile
- Main memory has a limited capacity

108. Diagram for operating system services:



109. Difference between a process and a program: a process is an instance of a running program while a program is a static set of instructions stored on disk.

110. Process termination: refers to the controlled ending of a program's execution in an operating system.

111. Process states and brief descriptions of each:

- New: process created and loaded into memory
- Ready: process waiting for the CPU
- Running: process actively executing instructions
- Blocked: process waiting for an event
- Terminated: process finished execution