Operating System

Ms. Swati Mali,

Ms.Nirmala Shinde Baloorkar

Assistant Professor

Department of Computer Engineering

Outline

- Introduction to Operating System
- Operating System Software
- Types of Architectures
- System Calls

Introduction to Operating System

- An Operating System (OS) is a program that manages the computer hardware.
- It also provides a basis for Application Programs and acts as an intermediary between computer user and computer hardware.
- Some widely used operating systems are

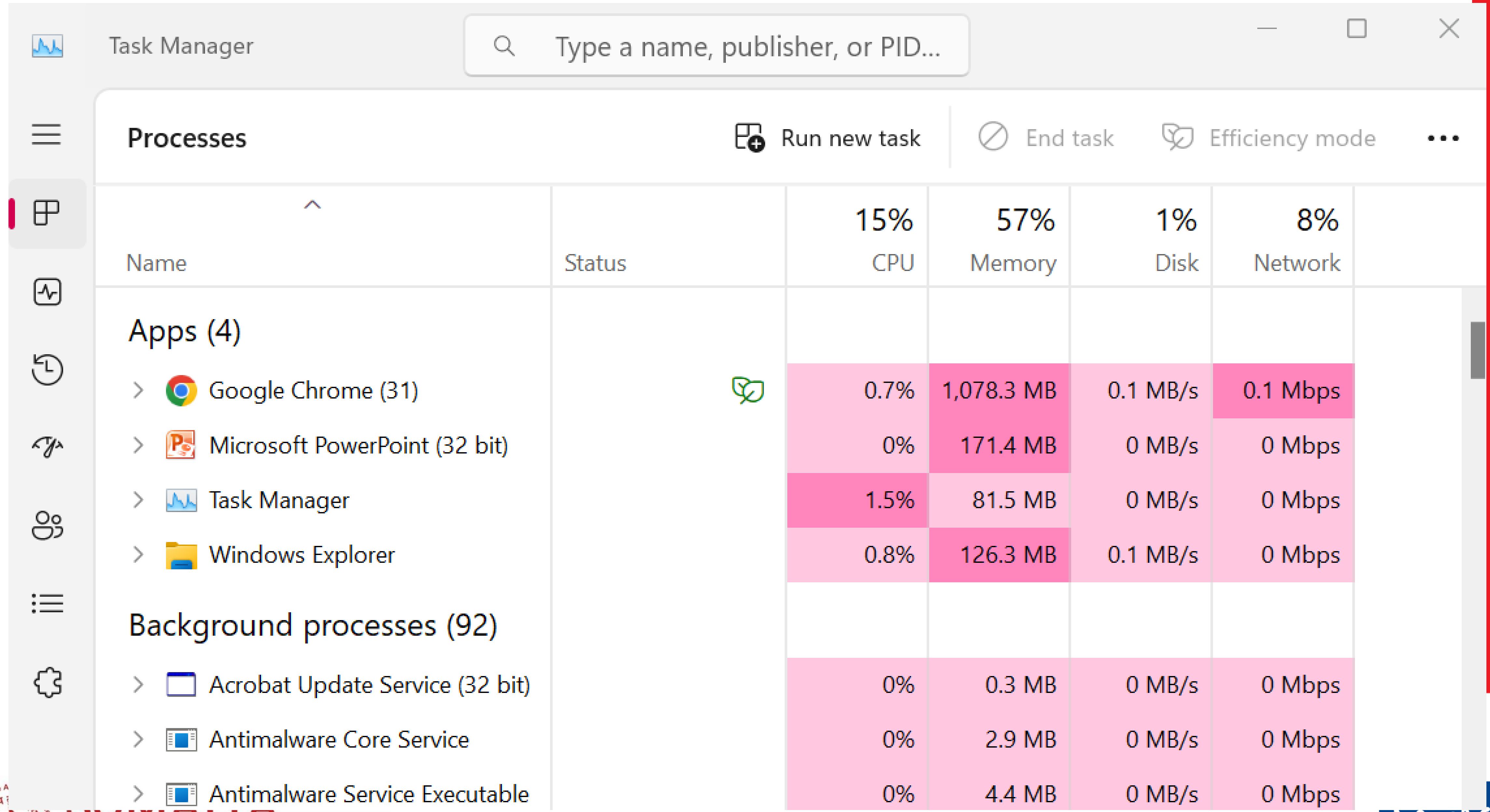
iOS
Linux
Microsoft Windows
Ubuntu

Basic Terminologies & Definitions

- Task
- Algorithm
- Program
- Process
- Thread

Basic Terminologies & Definitions

- Kernel
- Shell
- system call



Task Manager

Type a name, publisher, or PID...

Processes

Run new task | End task | Efficiency mode

Name	Status	15% CPU	57% Memory	1% Disk	8% Network
Apps (4)					
Google Chrome (31)		0.7%	1,078.3 MB	0.1 MB/s	0.1 Mbps
Microsoft PowerPoint (32 bit)		0%	171.4 MB	0 MB/s	0 Mbps
Task Manager		1.5%	81.5 MB	0 MB/s	0 Mbps
Windows Explorer		0.8%	126.3 MB	0.1 MB/s	0 Mbps
Background processes (92)					
Acrobat Update Service (32 bit)		0%	0.3 MB	0 MB/s	0 Mbps
Antimalware Core Service		0%	2.9 MB	0 MB/s	0 Mbps
Antimalware Service Executable		0%	4.4 MB	0 MB/s	0 Mbps

System Software

- A program that runs computer hardware and software.
- Examples: Operating System, BIOS, firmware
- Special Types of System Software: Translators, Compilers, Other Diagnostic tools
- Operates at the lowest computer level
- Manages & Operates Hardware
- Interface between Application Software and Hardware
- Synchronizes & controls data flow between memory, secondary storage device, Display, printers etc

Application software

- The softwares those run on system software to serve end users.
- Exemples: Animation, Graphics, Microsoft Office, etc.
- Developed to perform single or multiple tasks

Operating System

- Operating System is collection of software that controls hardware for scheduling the execution of other programs, mange storage; input, output & Communication Devices and the CPU.

Operating System

- Means of invoking Operating System functions is called System Calls.
- System calls:
  - Process Management,
  - File Management,
  - Time Management,
  - Memory and I/O Management

e.g. ch=fgetc(fp) is executed by

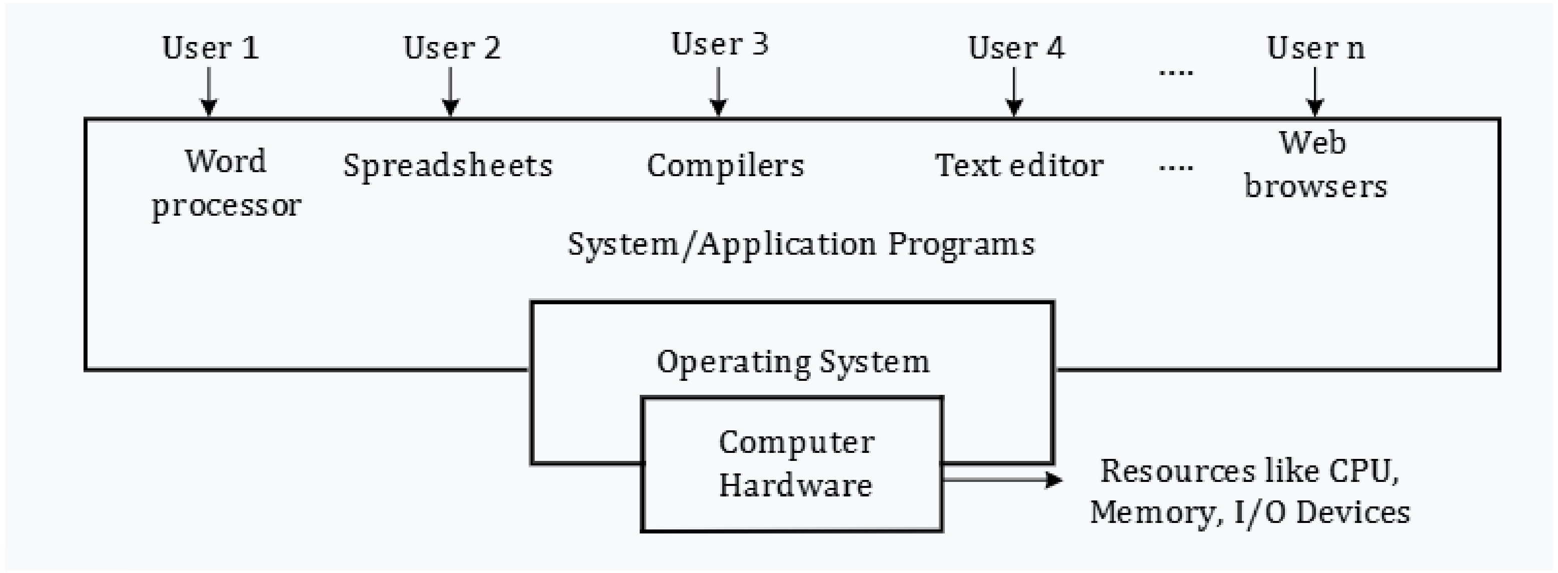
Count = read(fd, ch,1)

Elements of Computing system

- Processor: For execution of instructions.
- Memory Devices: For staring data and instructions (programs)
- Input /Output /Communication Devices: Used for communicating with the real world.
  - Receive the data from the user,
  - present data to the user
  - communicate with other devices.

Components of OS

1. The hardware
2. The OS
3. Application programs
4. Users



What does an OS do?

Resource Manager

- Convenience :
  - Operating System makes the communication process between the user and the hardware a lot easier, which is also required.
- Efficiency :
  - By efficiency, we mean the suitable allocation of resources and the quality of processing of data. So an operating system should provide an efficient usage of the system.

Quiz

Which of the following statement best describes an operating system?

A. It is a type of antivirus software.

B. It is a programming language used to develop application.

C. It is a software that manages computer hardware and software resources.

D. It is a collection of hardware components in a computer system.

Quiz

Which of the following is NOT a function performed by an operating system?

A. File System Management

B. Data Encryption

C. User Interface

D. Memory Management

Quiz

Which of the following modes of CPU execution provides the highest level of privilege?

A. Protected Mode

B. Supervisor Mode

C. Kernel Mode

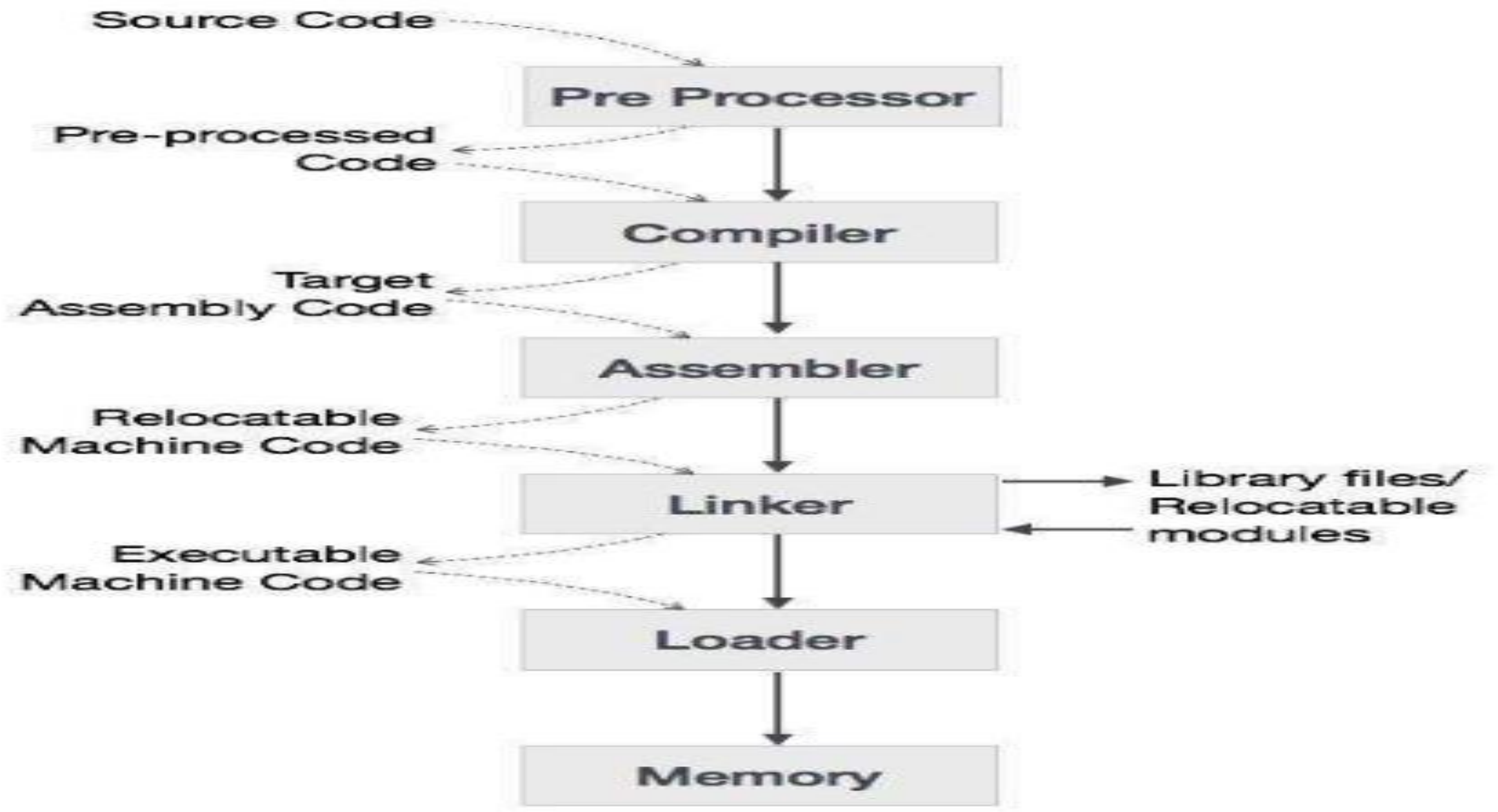
D. User Mode

System programs

- **Device Drivers**:
  - Facilitate communication between the OS and hardware devices.
  - Provide necessary instructions for the OS to control hardware components.
- **Macro Processors**:
  - Expand macros within the source code if macros are used.
  - Replace macro instructions with their corresponding sequences of code before compilation.
- **Compilers**:
  - Translate high-level source code into assembly language or machine code.
  - Produce object code (usually stored in object files).
- **Assemblers**:
  - Convert assembly language code into machine code.
  - Generate object files containing machine code.

System programs

- **Linkers**:
  - Combine multiple object files into a single executable file.
  - Resolve symbol references between object files and link necessary libraries.
- **Loaders**:
  - Load the executable file into memory for execution.
  - Allocate memory and set up the execution environment.
- **Interpreters**:
  - Execute high-level programming code directly, line-by-line.
  - Translate and execute code simultaneously without producing an intermediate machine code file.
  - Interpreters can be used instead of compilers for certain languages (like Python).


Source Code
Pre Processor
Pre-processed Code
Compiler
Target Assembly Code
Assembler
Relocatable Machine Code
Linker
Library files/ Relocatable modules
Executable Machine Code
Loader
Memory

How a program executes?

```
#include <stdio.h>
int main() {
    int a = 5;
    int b = 7;
    int sum = a + b;
    printf("Sum of %d and %d is %d\n", a, b, sum);
    return 0;
}
```

How a program executes?

- Macroprocessor / **Preprocessing**:
  - The preprocessor expands directives such as #include <stdio.h>.
  - It includes the contents of stdio.h and expands any macros defined within it. E.g. #define pi 3.142

➔ Output after Preprocessing:

- Expanded source code including contents of stdio.h.

- Example: Expanded printf function declaration and macros from stdio.h.

How a program executes?

2 Compilation:

- The compiler translates the C code into assembly language or directly into machine code specific to the target platform.

- Output after Compilation (example):
  - Object code (machine code) instructions.
  - Symbol table (example): ```

```
Symbol Table:
  ---------------------
  | Symbol | Address |
  ---------------------
  |   main |  0000H  |
  |      a |  0001H  |
  |      b |  0002H  |
  |    sum |  0003H  |
  ---------------------
  ```
```

- Example (simplified): Machine code instructions for setting up variables `a`, `b`, calculating `sum`, and calling `printf`.
```

How a program executes?

3. Assembly (if applicable):

- If assembly code is a readable representation of the machine code instructions.

Example Assembly Language Code

```
; Data Section
        ORG 1000H
a       DB 05H    ; Define variable 'a' with initial value 05H
b       DB 07H    ; Define variable 'b' with initial value 07H
sum     DS 01H    ; Define space for result 'sum'

; Code Section
        ORG 2000H
START:
        MOV A, a    ; Move value of 'a' into accumulator A
        ADD A, b    ; Add value of 'b' to accumulator A
        MOV sum, A  ; Store result in 'sum'
        HLT         ; Halt the program
        END START   ; End of program
```

How a program executes?

- Linker
  - combines the compiled object code with necessary library functions (like printf from stdio.h) into a single executable file
  - performs address calculations and relocation of code and data sections to ensure proper execution in the target memory space.
  - It adjusts addresses as necessary to avoid conflicts and ensure correct operation.

How a program executes?

- Loader
  - The input to the loader is executable file generated by the linker.
  - contains machine code instructions, data, and information about memory layout.
  - The loader loads its contents into memory.
  - It allocates memory space for the program's code, data segments, stack, and heap as necessary.
  - resolves addresses and relocations specified by the linker.
  - adjusts addresses in the executable file to reflect the actual memory addresses where the program will reside during execution.

How a program executes?

Memory Layout:

```
---------------------------------------------------
| Code Segment   | Data Segment   | Stack            |
| (Executable)      |(Initialized)        | (Function        |
|                                  | (Variables)    | Call Stack)  |
---------------------------------------------------
```

COMPILER	INTERPRETER
A compiler is a program which coverts the entire source code of a programming language into executable machine code for a CPU.	interpreter takes a source program and runs it line by line, translating each line as it comes to it.
Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present any where in the program.	Its Debugging is easier as it continues translating the program until the error is met
Generates intermediate object code.	No intermediate object code is generated.
Examples: C, C++, Java	Examples: Python, Perl

Quiz

GATE CS 1998 | Question 25

In a resident- OS computer, which of the following system software must reside in the main memory under all situations?
(A) Assembler
(B) Linker
(C) Loader
(D) Compiler

Quiz

A linker program

A. a. places the program in the memory for the purpose of execution.

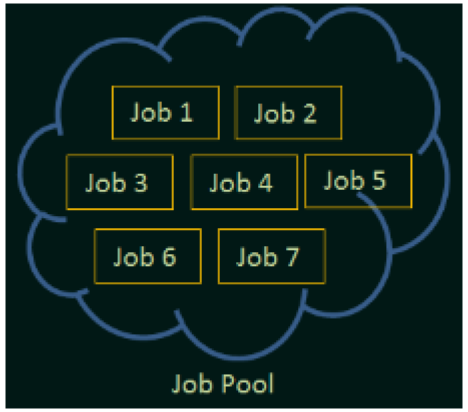
B. b. relocates the program to execute from the specific memory area allocated to it.

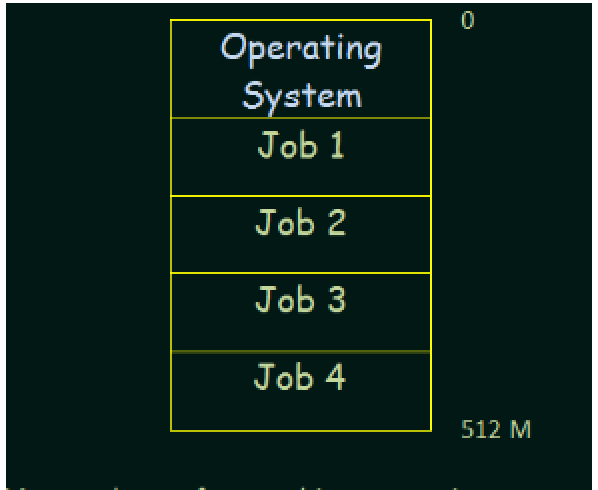
C. c. links the program with other programs needed for its execution.

D. d. interfaces the program with the entities generating its input data.

Operating System Structure

- Multiprogramming
  - A single user can't in general, keep either the **CPU or the I/O devices busy at all times**.
  - Multiprogramming increases CPU utilization by **organizing jobs (code & data) so that the CPU always has one to execute**.





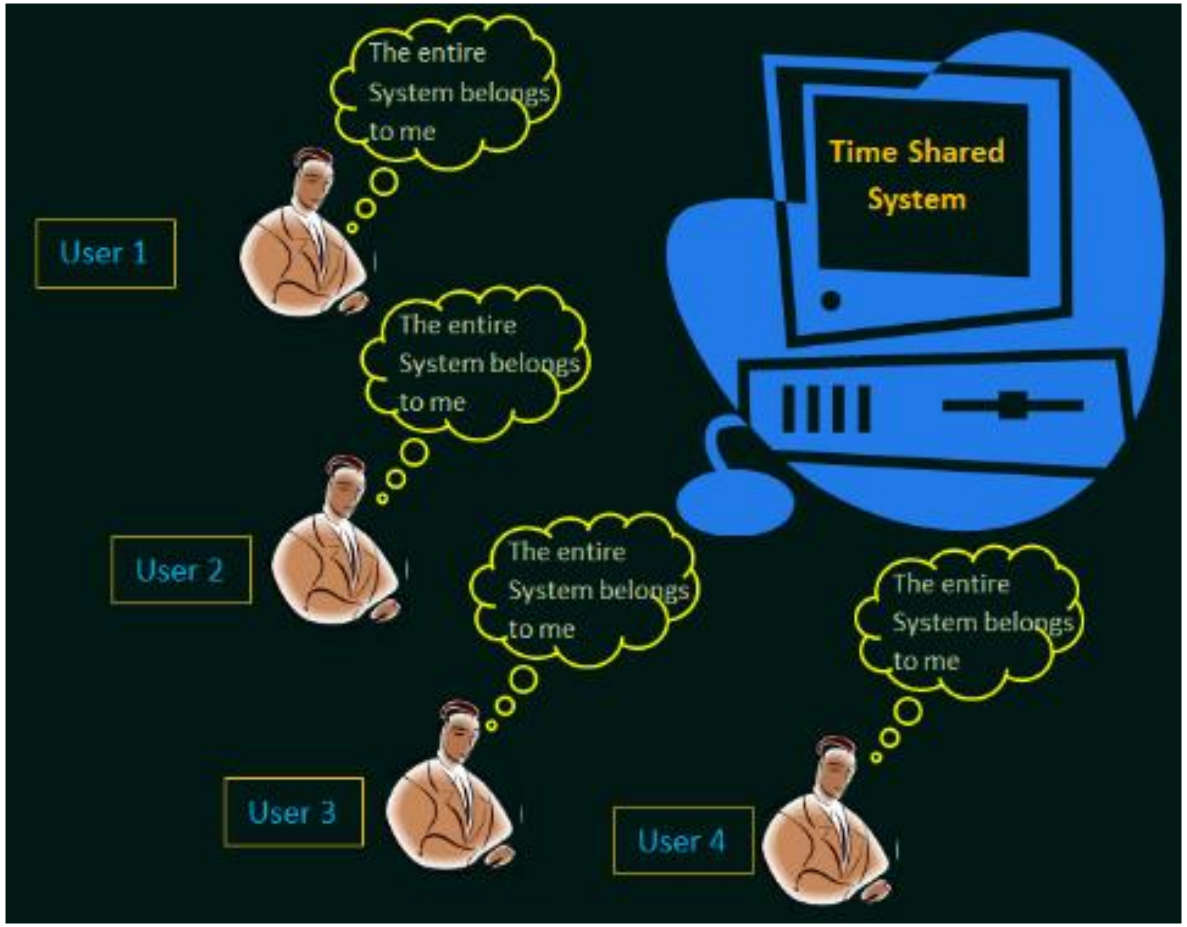
Memory layout for a multiprogramming system

Operating System Structure (cont…)

- Time Sharing (Multitasking)
  - CPU executes multiple jobs by switching among them
  - Switches occur so frequently that the users can interact with each program while it is running
  - Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system.
  - A time-shared operating system allows many users to share the computer simultaneously.

Operating System Structure (cont…)

- Uses CPU scheduling & multiprogramming to provide each user with small portion of a time-shared computer.
- Each user has at least one separate program in memory.
- A program loaded into memory and executing is a called a "Process".


The entire System belongs to me
Time Shared System
User 1
The entire System belongs to me
User 2
The entire System belongs to me
The entire System belongs to me
User 3
User 4

Evolution of OS

1. **Early Systems (1940s-1950s) - Batch Processing Systems**:
   - These systems processed one job at a time in a batch mode.
   - Jobs were collected and executed sequentially without user interaction.
   - Examples: IBM 701, UNIVAC.
2. **Simple Batch Systems (1950s-1960s) - Resident Monitor:**
   - An early form of the OS that resided in memory to handle job sequencing.
   - Jobs were processed in batches with minimal manual intervention.
   - Examples: IBM 1401.
3. **Single-User, Single-Tasking Systems (1960s-1970s):**
4. **Single-User, Multiprocessing Systems (1970s-1980s):**
   - Examples: Early versions of macOS (Classic Mac OS), MS-DOS with TSR (Terminate and Stay Resident) programs.

Evolution of OS (cont…)

5. **Multiprogramming Systems (1960s):**
   5. Allowed multiple programs to be loaded into memory and executed concurrently by the CPU.
   6. Examples: IBM System/360, CTSS (Compatible Time-Sharing System).
6. **Time-Sharing Systems (1960s-1970s):**
   5. Examples: MULTICS (Multiplexed Information and Computing Service), Unix.
7. **Multi-User, Multiprocessing Systems (1970s-1980s):**
   5. Examples: Unix, VMS (Virtual Memory System).
8. **Personal Computer Operating Systems (1980s-1990s) - Single-User, Multiprocessing:**
   5. Graphical User Interfaces (GUIs) gained popularity, making computers more accessible.
   6. Examples: Windows 95, Mac OS.

Evolution of OS (cont…)

9. **Network Operating Systems (1980s-1990s)**:
   9. Enabled file sharing, printer sharing, and inter-process communication over networks.
   10. Examples: Novell NetWare, Windows NT, early Unix-based systems with networking extensions.
10. **Distributed Operating Systems (1990s):**
    9. Managed independent computers as a single coherent system.
    10. Examples: Amoeba, Plan 9, early versions of Linux with distributed capabilities.
11. **Modern Operating Systems (2000s-Present) - Multi-User, Multiprocessing Systems:**
    - Support advanced multitasking, multi-user environments, and extensive networking.
    - Emphasize security, stability, and user-friendly interfaces.
    - Examples: Modern Windows, macOS, Linux distributions, Android, iOS.

Evolution of OS (cont…)

12. Mobile Operating Systems (2000s-Present):

12. Designed for mobile devices with touch interfaces, efficient power management, and connectivity.
13. Emphasize app ecosystems and seamless user experiences.
14. Examples: iOS, Android.

13. Cloud and Virtualization (2010s-Present):

- Examples: VMware, Hyper-V, Kubernetes, cloud-based OS like Google Chrome OS.

Types of Architectures - Operating System

- Monolithic Architecture
- Layered Architecture
- Micro-Kernel Architecture
- Hybrid Architecture

Monolithic Architecture

- Each component of the **operating system is contained in the kernel** and the components of the operating system **communicate with each other using function calls**.
- E.g. OS/360, VMX, and LINUX.

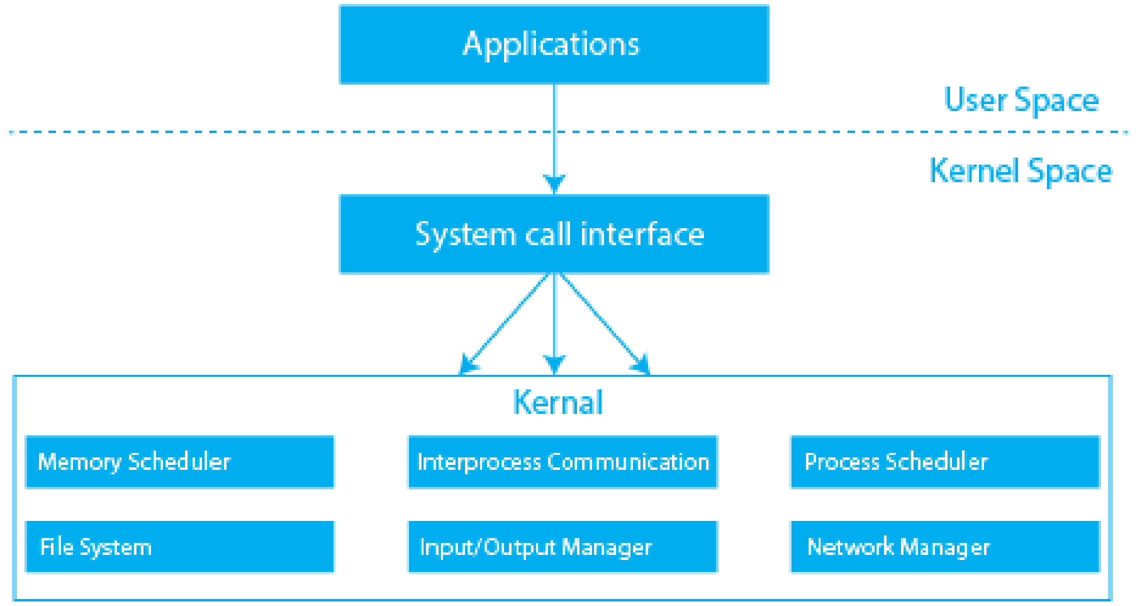


Image source : [Operating System Architecture (prepbytes.com)]

Monolithic Architecture (cont…)

- **Performance**:
  - High due to direct function calls within the kernel.
- **Modularity**:
  - Low, as all components are intertwined in a single large codebase.
- **Maintenance**:
  - Difficult, because changes in one part can affect many others.
- **Security**:
  - Lower, because a bug in any part of the kernel can compromise the entire system.

Layered Architecture

- The OS is separated into layers or levels in this kind of arrangement.
- Layer 0 (the lowest layer) contains the hardware, and layer 1 (the highest layer) contains the user interface (layer N).
- These layers are organized hierarchically, with the top-level layers making use of the capabilities of the lower-level ones.
- E.g. Windows XP, and LINUX

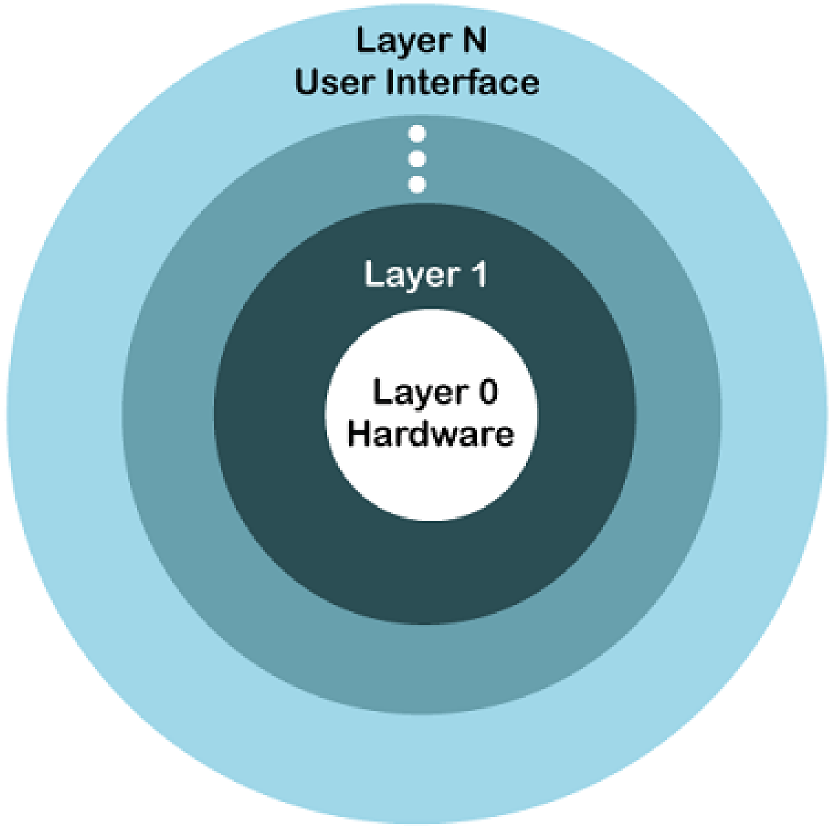
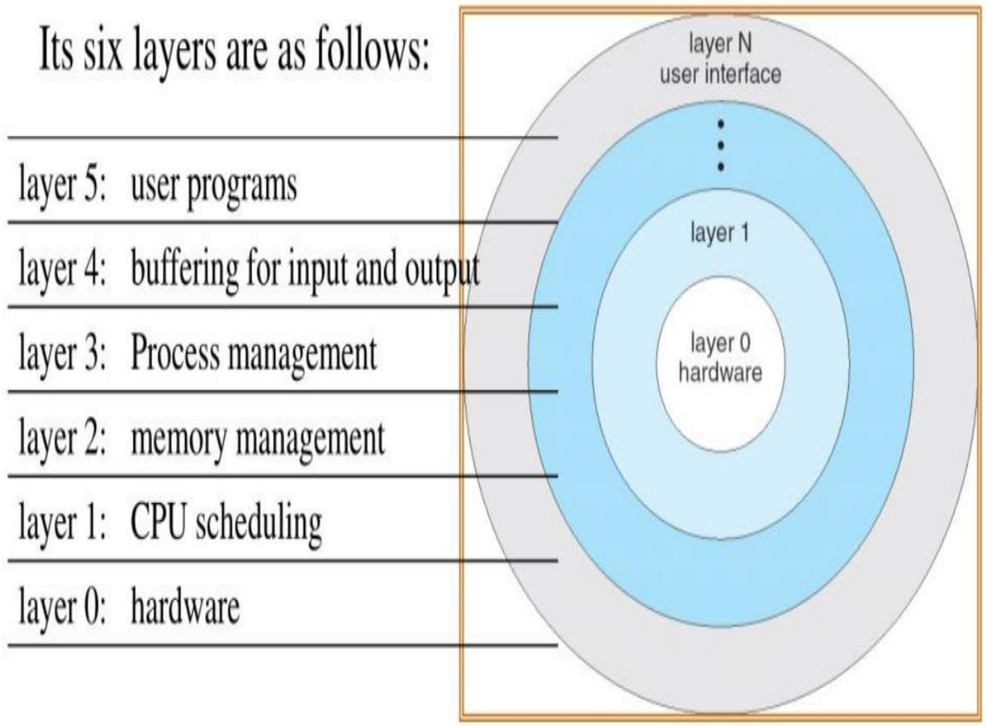


Image source : Operating System Structure - javatpoint

Layered Architecture (cont…)

- **Performance**:
  - Moderate, as each layer adds overhead.
- **Modularity**:
  - Higher than monolithic, since each layer has specific functionality.
- **Maintenance**:
  - Easier than monolithic, as changes are localized within layers.
- **Security**:
  - Improved compared to monolithic, but a bug in lower layers can still affect upper layers.



Its six layers are as follows:

layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	Process management
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

Image source : Information to know about Operating System – Programmer Prodigy (code.blog)

Microkernel Architecture

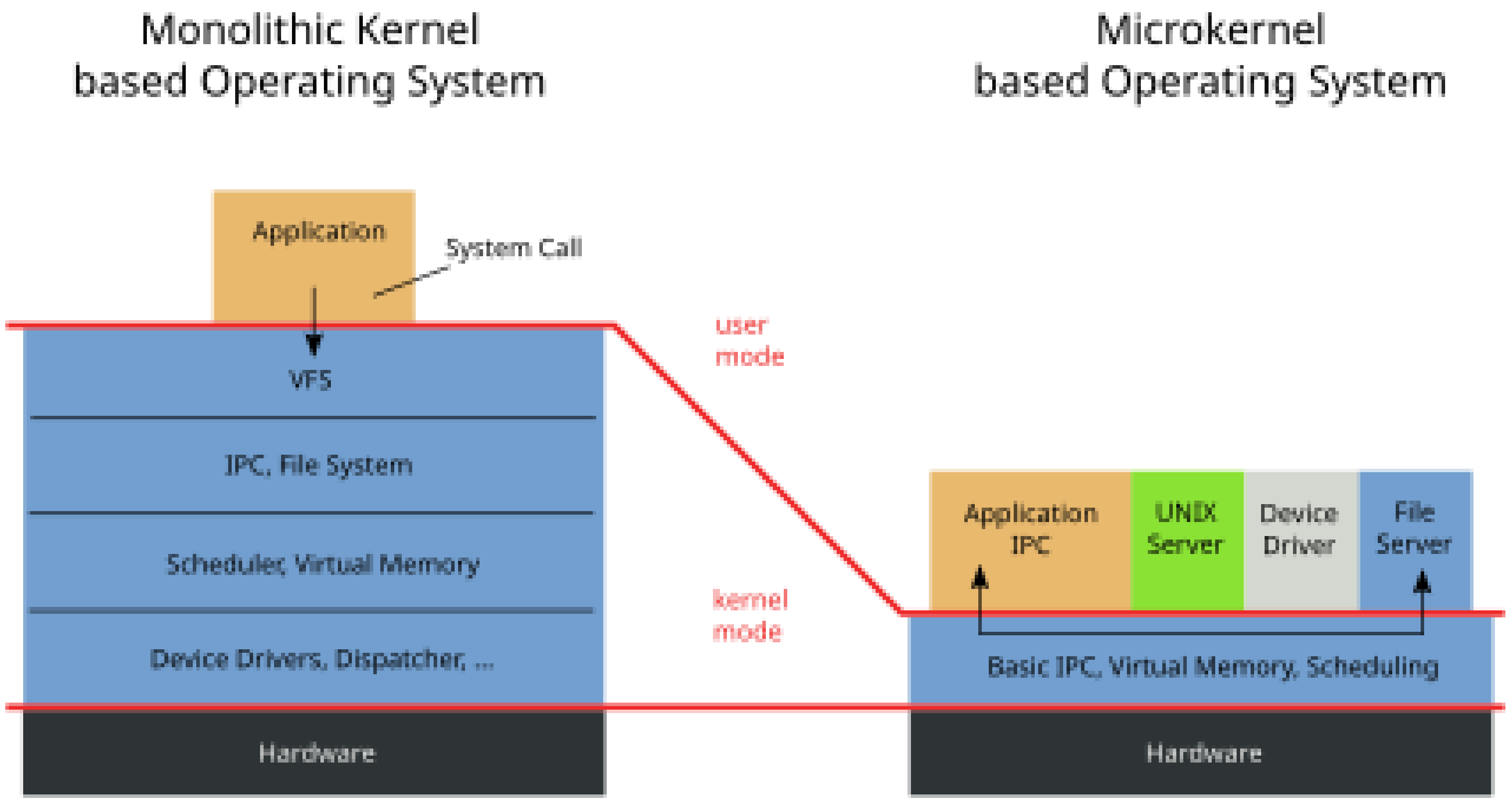

Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Image source: Wikipedia

Microkernel Architecture

- **Performance**:
  - Lower due to more context switches and inter-process communication.
- **Modularity**:
  - Very high, as only essential services are in the kernel and others run in user space.
- **Maintenance**:
  - Easiest, as most services are user-space programs that can be updated independently.
- **Security**:
  - Highest, because faults in user-space services do not affect the kernel.

Microkernel vs Monolithic

S. No.	Parameters	Microkernel	Monolithic kernel
1.	**Address Space**	In microkernel, user services and kernel services are kept in separate address space.	In monolithic kernel, both user services and kernel services are kept in the same address space.
2.	**Design and Implementation**	OS is complex to design.	OS is easy to design and implement.
3.	**Size**	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.
4.	**Functionality**	Easier to add new functionalities.	Difficult to add new functionalities.
5.	**Coding**	To design a microkernel, more code is required.	Less code when compared to microkernel

Microkernel vs Monolithic (cont…)

S. No.	Parameters	Microkernel	Monolithic kernel
6.	**Failure**	Failure of one component does not effect the working of micro kernel.	Failure of one component in a monolithic kernel leads to the failure of the entire system.
7.	**Processing Speed**	Execution speed is low.	Execution speed is high.
8.	**Extend**	It is easy to extend Microkernel.	It is not easy to extend monolithic kernel.
9.	**Communication**	To implement IPC messaging queues are used by the communication microkernels.	Signals and Sockets are utilized to implement IPC in monolithic kernels.
10.	**Debugging**	Debugging is simple.	Debugging is difficult.

Microkernel vs Monolithic (cont…)

S. No.	Parameters	Microkernel	Monolithic kernel
11.	**Maintain**	It is simple to maintain.	Extra time and resources are needed for maintenance.
12.	**Message passing and Context switching**	Message forwarding and context switching are required by the microkernel.	Message passing and context switching are not required while the kernel is working.
13.	**Services**	The kernel only offers IPC and low-level device management services.	The Kernel contains all of the operating system's services.
14.	**Example**	**Example :** Mac OS X.	**Example :** DOS, classical early versions of BSD, Unix, Solaris, Mac OS, etc.

Hybrid Architecture

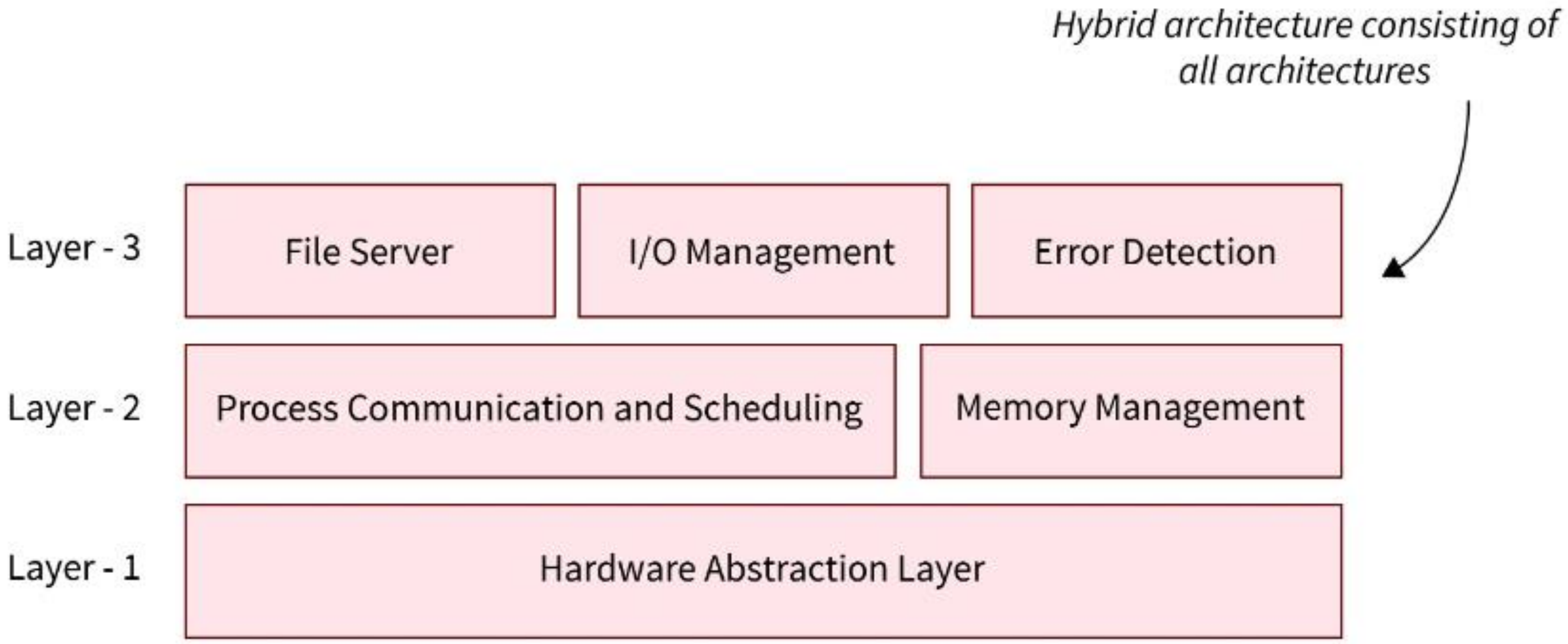

Hybrid architecture consisting of all architectures
Layer - 3
File Server
I/O Management
Error Detection
Layer - 2
Process Communication and Scheduling
Memory Management
Layer - 1
Hardware Abstraction Layer

Image source: Architecture of Operating System - Scaler Topics

Hybrid Architecture (cont…)

- **Performance:**
  - They allow various architectural components to provide specialized services.
  - This flexibility can lead to better overall system performance.
- **Modularity:**
  - Each layer focuses on specific functionality (e.g., file system, memory management).
  - Developers can work on individual layers independently.
- **Maintenance:**
  - Easier maintenance due to clear module boundaries.
  - Updates or bug fixes can be applied to specific layers without affecting the entire system.
- **Security:**
  - Separating critical services (kernel space) from less critical ones (user space) enhances security.
  - Kernel-level services are protected from user-level code.

Current OS designs

- **Windows NT and successors (2000, XP, Vista, 7, 8, 10, 11)**: Hybrid kernel with microkernel elements.
- **macOS (and iOS, iPadOS, watchOS, tvOS)**: Based on the XNU kernel, combining Mach microkernel and FreeBSD components.
- **Linux distributions**: Monolithic kernel with dynamically loadable modules, offering hybrid-like flexibility.
- **Android**: Built on the Linux kernel, following a similar modular approach.
- **Solaris**: Primarily monolithic but incorporates some microkernel principles with loadable kernel modules.

Quiz

In which among the given architecture executes different components of an operating system separately?

A. Monolithic Architecture

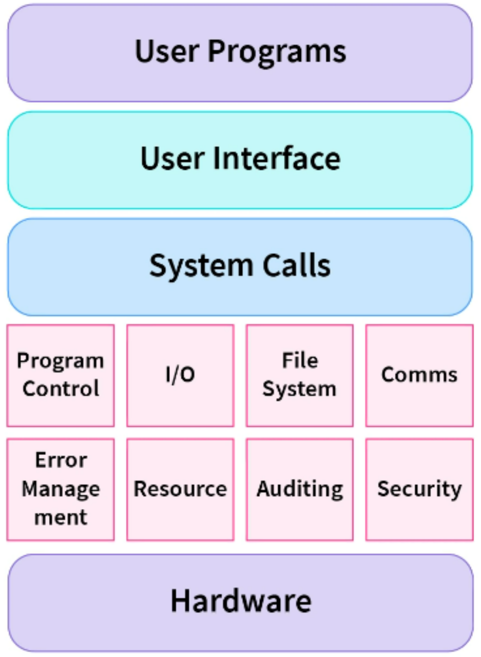
B. Layered Architecture

C. Microkernel Architecture

D. None of the given

System Calls

- System calls provide an interface to the services made available by an Operating System.
- Kernel is the core component of the Operating System that has complete control of the hardware.



System calls

- Working of system call

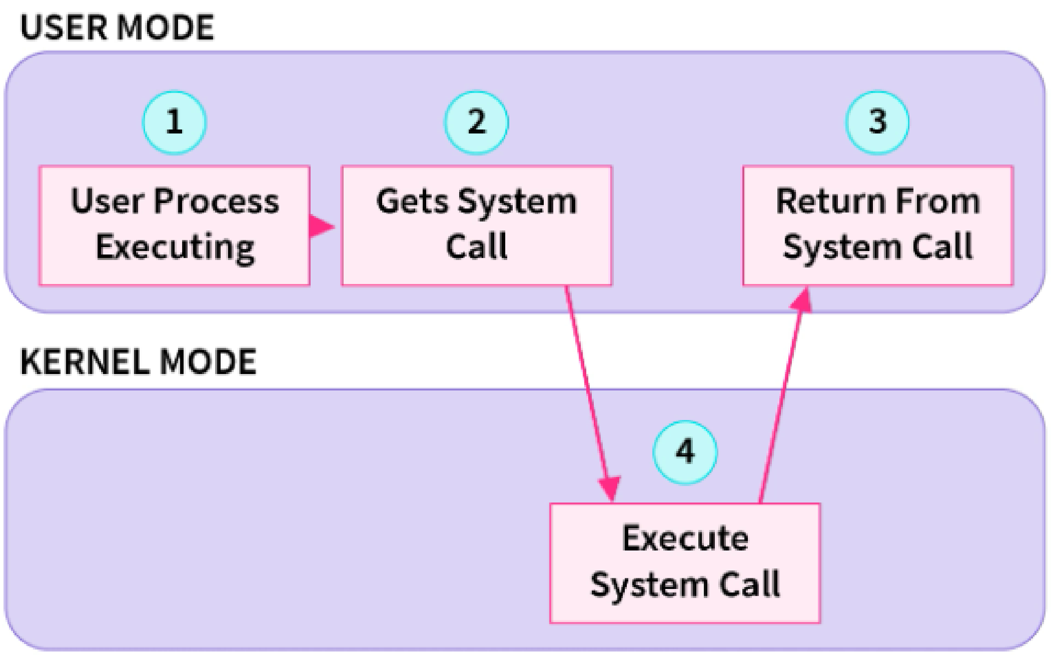
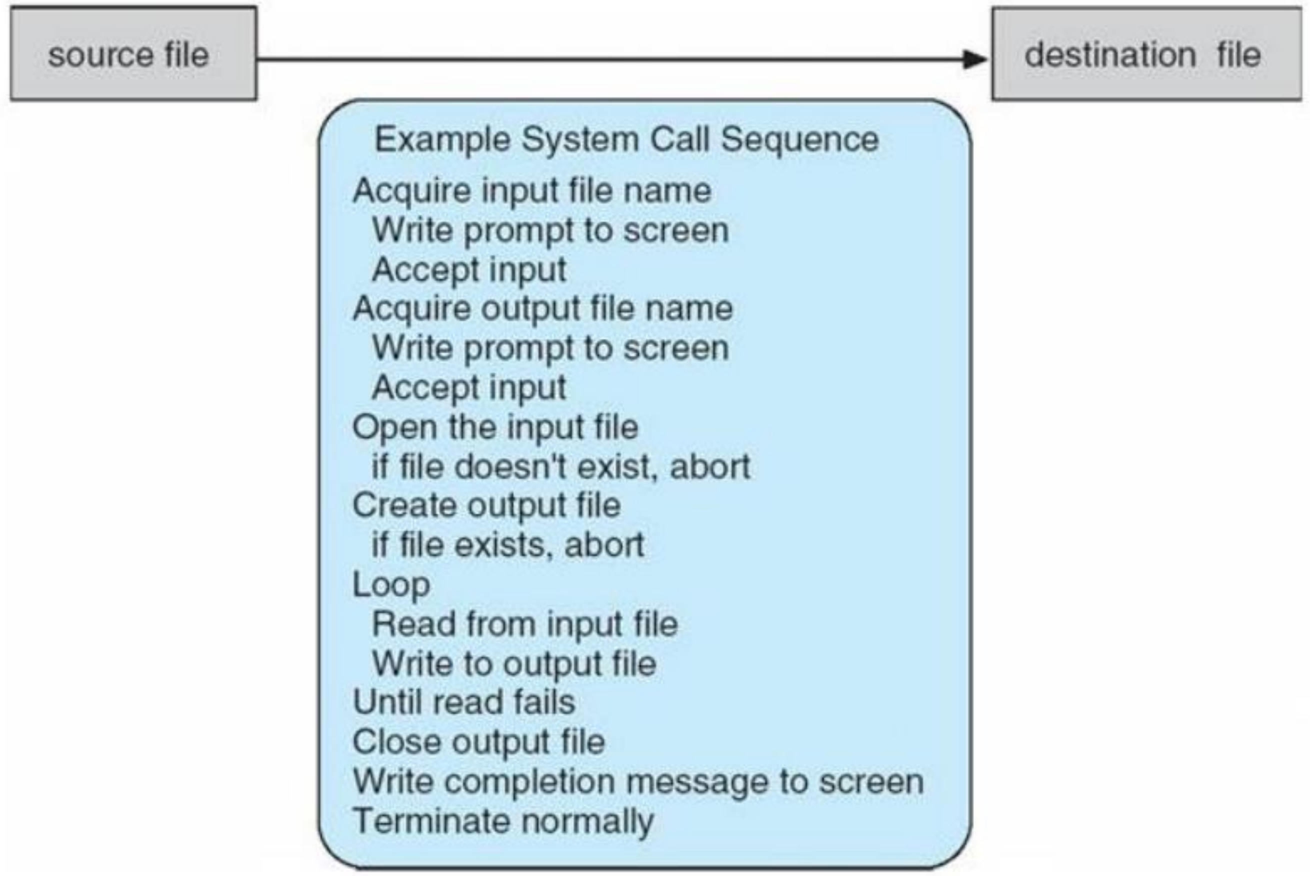


Image source: System Calls in OS (Operating System) - Scaler Topics

Example of System Call

Read data from one file and copy them to another file.



```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Image source: [System Calls (iit.edu)]

Types of System Calls

- Process Control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File Management
  - Create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device Management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information/Time Management
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communication
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Types of System Calls

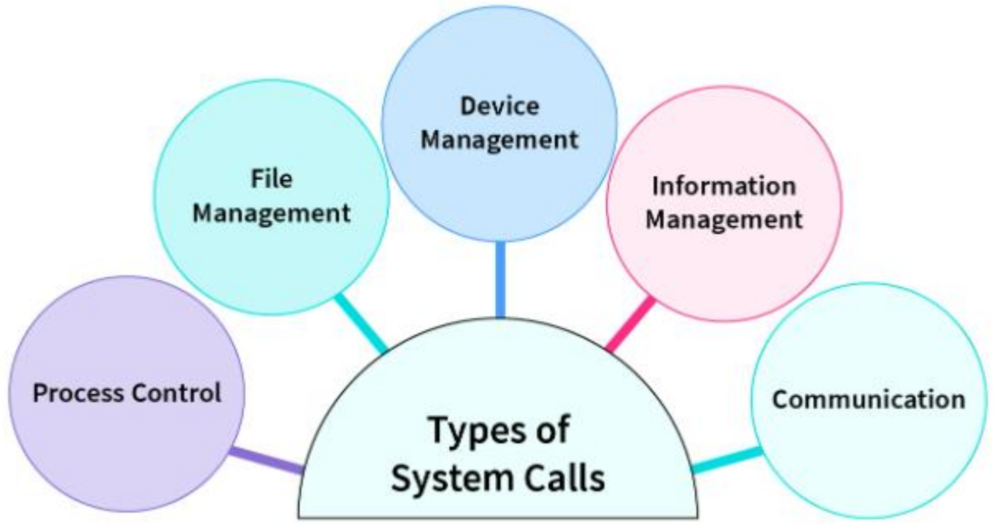


Image source: System Calls in OS (Operating System) - Scaler Topics

Process System Calls

- **fork()**:
  - Creates a new process (child process) that runs concurrently with the parent process.
- **exec()**:
  - Loads a new program into the current process space, replacing the current process with the new program.
- **wait() / waitpid()**:
  - Suspends execution of the calling process until one of its child processes terminates. waitpid() allows more control over which child process to wait for.
- **exit()**:
  - Terminates the calling process and optionally returns an exit status to the operating system.
- **getpid()**:
  - Returns the process ID (PID) of the current process.
- **getppid()**:
  - Returns the PID of the parent process.
- **kill()**:
  - Terminates a specified process.

File System Calls

- open()
- close()
- read()
- write()
- lseek()
- mkdir()
- rmdir()
- unlink()
- rename()
- stat() / fstat()
- chmod()
- chown()
- Link()
- Unlink()
- Mount()
- Umount()

Information Management System Calls

- time()
- gettimeofday()
- clock_gettime()
- alarm()
- sleep()
- nanosleep()
- getitimer()
- setitimer()
- Uptime()

Memory System Calls

- **brk() / sbrk()**:
  - Adjusts the location of the program break, which specifies the end of the process's data segment (heap)
- **mmap() / munmap()**:
  - Maps or unmaps files or devices into memory.
- **malloc() / calloc() / realloc() / free()**:
  - Allocates or deallocates dynamic memory.

I/O management related system calls

- **open() / close()**:
  - Opens or closes a file or device.
- **read() / write()**:
  - Reads from or writes to a file descriptor.
- **ioctl()**:
  - Performs device-specific I/O operations.

Communication System Calls

- Pipe()
- Shmget()
- Mmap()

System Boot

- The boot process is something that
  - happens every time
  - you turn your computer on.
  - You don’t really see it,
  - because it happens so fast.
- You press the power button,
  - come back a few minutes later and
  - Windows XP, or Windows 10, or whatever Operating System you use
  - is all loaded.

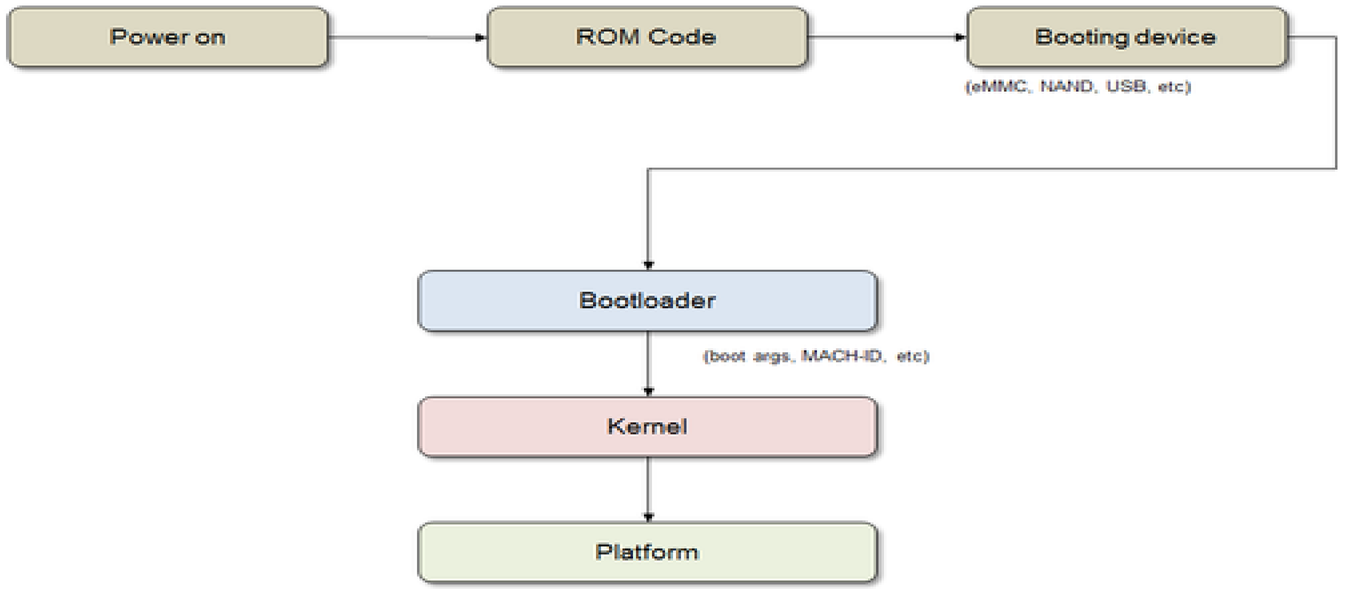
System Boot

- When power is initialized on system,
    - execution starts at a fixed memory location,
    - **Firmware ROM used to hold initial boot code**

- Operating system must be
    - made available to hardware
    - so hardware can start it

System Boot (cont…)

- Sometimes two-step process where
  - **Boot block** at fixed location loaded by ROM code,
  - which loads bootstrap program from disk
  - Then loader continues with its job



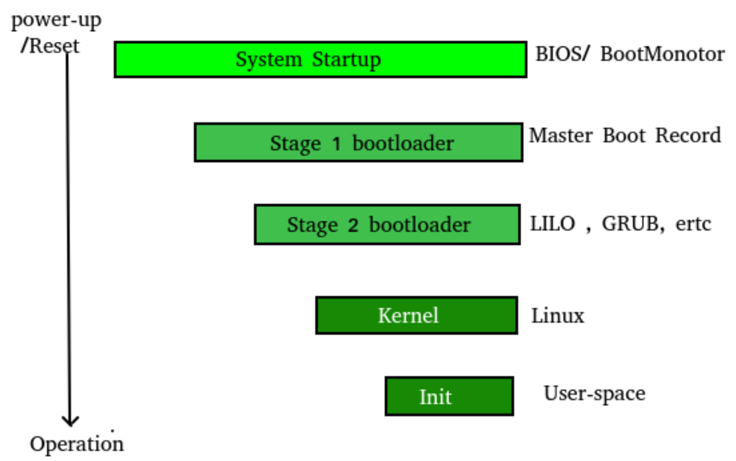
System Boot (cont…)

- **Bootstrap loader**
  - Small piece of code,
  - stored in **ROM** or **EEPROM** (electrically erasable programmable read-only memory)
  - locates the kernel,
  - loads it into memory, and
  - starts it

System Boot (cont…)

- Common bootstrap loader,
  - **GRUB**, (the **GRand Unified Bootloader)**
  - allows selection of kernel from
    - multiple disks,
    - versions,
    - kernel options

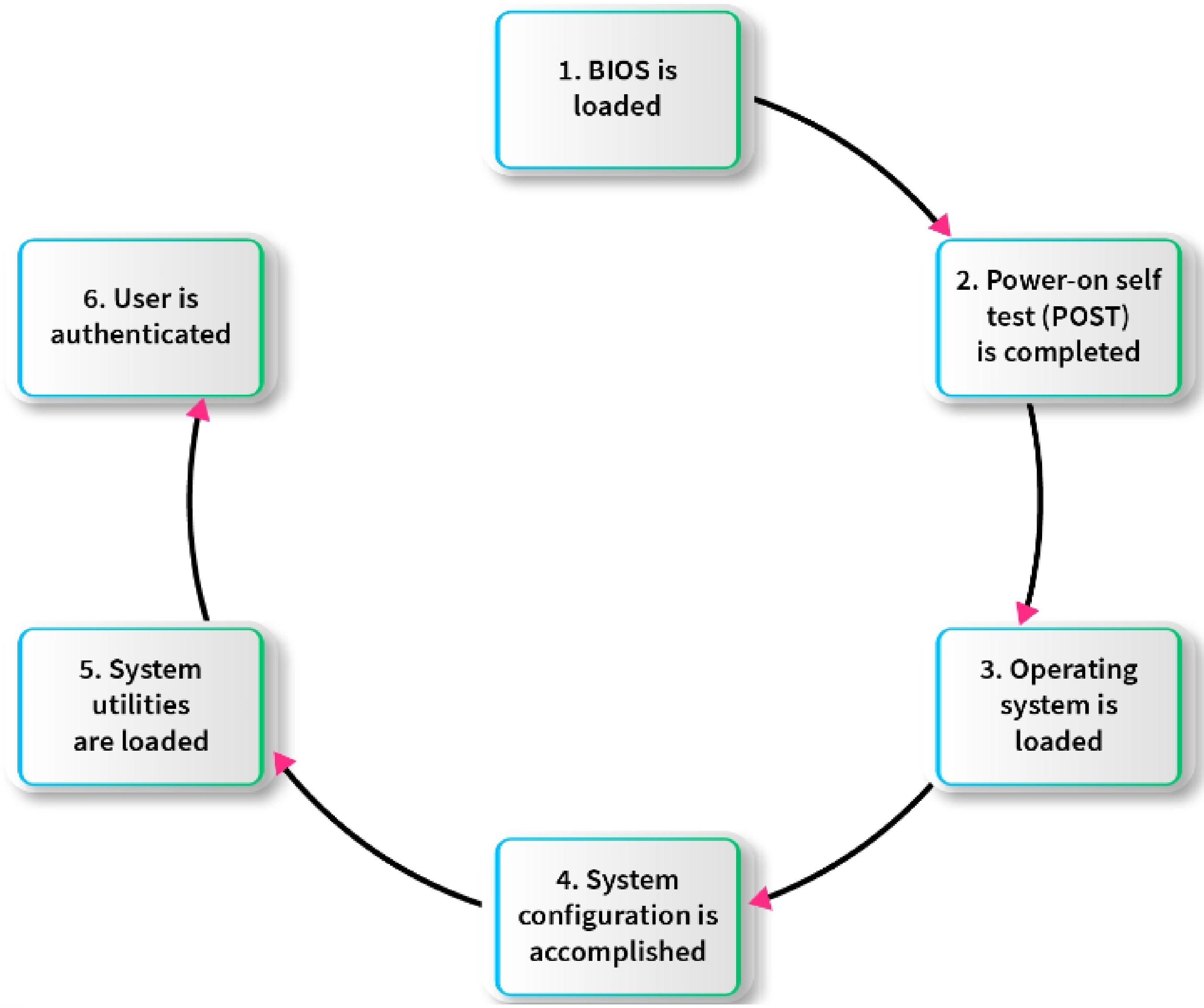
- Kernel loads and
  - system is then **running**



Types of Booting

- Cold Booting/ Hard Booting:
  - Cold booting is the process when our computer system moves from the shutdown state to the start by pressing the power button.
  - The system reads the BIOS from ROM and will eventually load the Operating System.
- Warm Booting/ Soft Booting:
  - Warm booting is the process in which the computer gets restarted due to reasons like setting the configuration for newly installed software or hardware. Warm booting is called as rebooting.

Booting Process



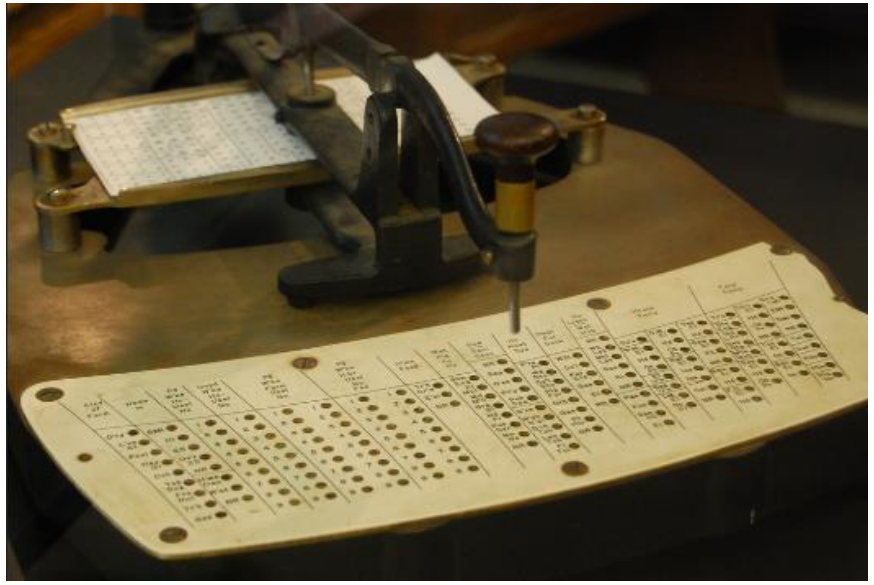
1. BIOS is loaded
2. Power-on self test (POST) is completed
3. Operating system is loaded
4. System configuration is accomplished
5. System utilities are loaded
6. User is authenticated

Booting

- **Power-On Self-Test (POST)**:
  - hardware (CPU, memory, motherboard, etc.) performs a Power-On Self-Test (POST).
  - also identifies connected devices like keyboards, monitors, and storage drives.
- **BIOS/UEFI Initialization**:
  - BIOS/UEFI is firmware stored on motherboard that provides basic control and communication between the operating system and hardware components.
- **Boot Loader Stage**:
  - The BIOS/UEFI locates and loads the boot loader program from the storage device (typically the hard drive or SSD).
  - The boot loader (such as GRUB in Linux or NTLDR in older versions of Windows) initializes the operating system.
- **System Operation**: Once logged in, users can interact with the operating system, run applications, and perform tasks using the system's resources.

- **Kernel Initialization**:
  - The boot loader loads the operating system's kernel into memory.
  - The kernel is the core of the operating system that manages system resources, such as memory, CPU, and devices.
- **Init Process**:
  - kernel once loaded, starts the init process
  - The init process is the first user-space process that initializes the system environment and starts essential system services.
- **User Login/Graphical Interface**:
  - system prompts for user login (in text mode) or launches the graphical user interface (GUI).
  - User login allows users to access the system and start applications.

650-2
gettyimages
H. Armstrong Roberts/ClassicStock
1159414065

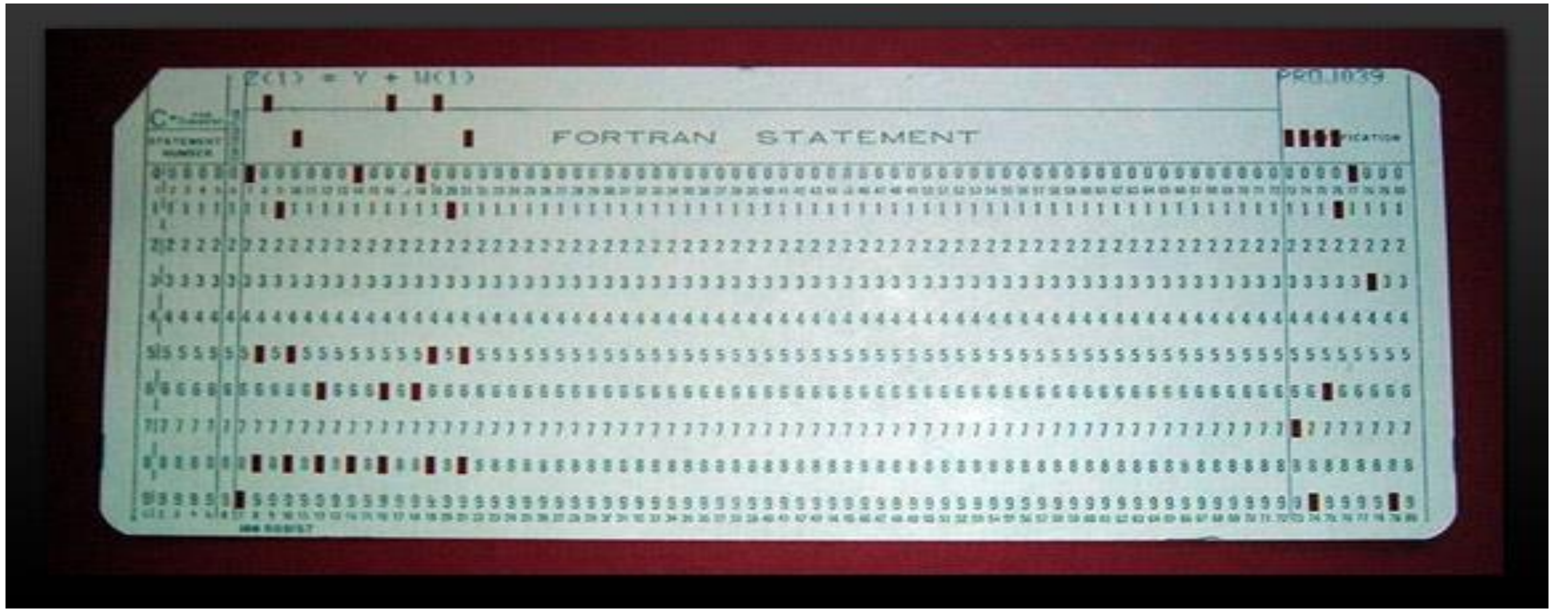


??

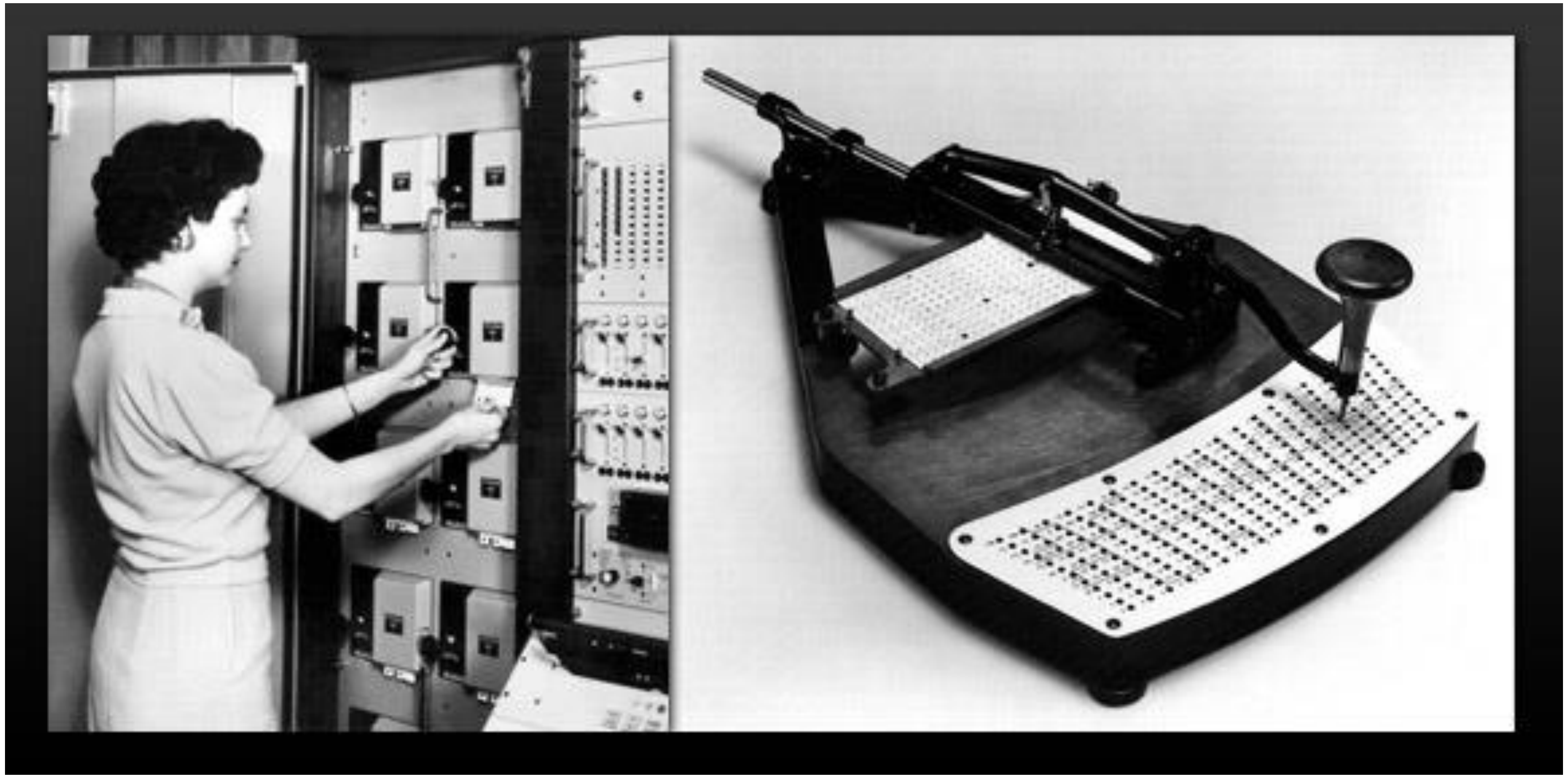
IMATION
MAGNUS™ 1.2
DATA CARTRIDGE / CARTOUCHE DE DONNEES
SLR

TDK
DATA CARTRIDGE
DC-6250
Cartouche Numérique
DataLife
DDS4
FUJIFILM
DG4-150

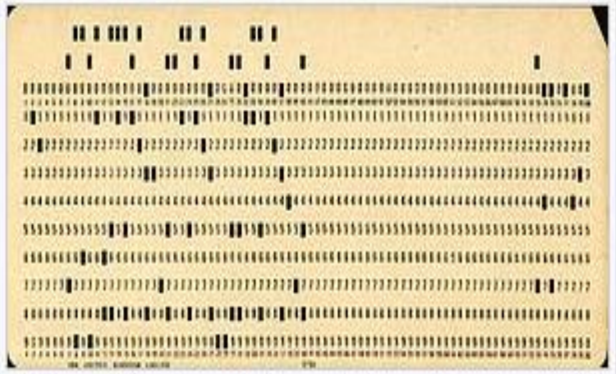
Punch card/A punched card



Punch card/A punched card



Punch card Reader, A punched card

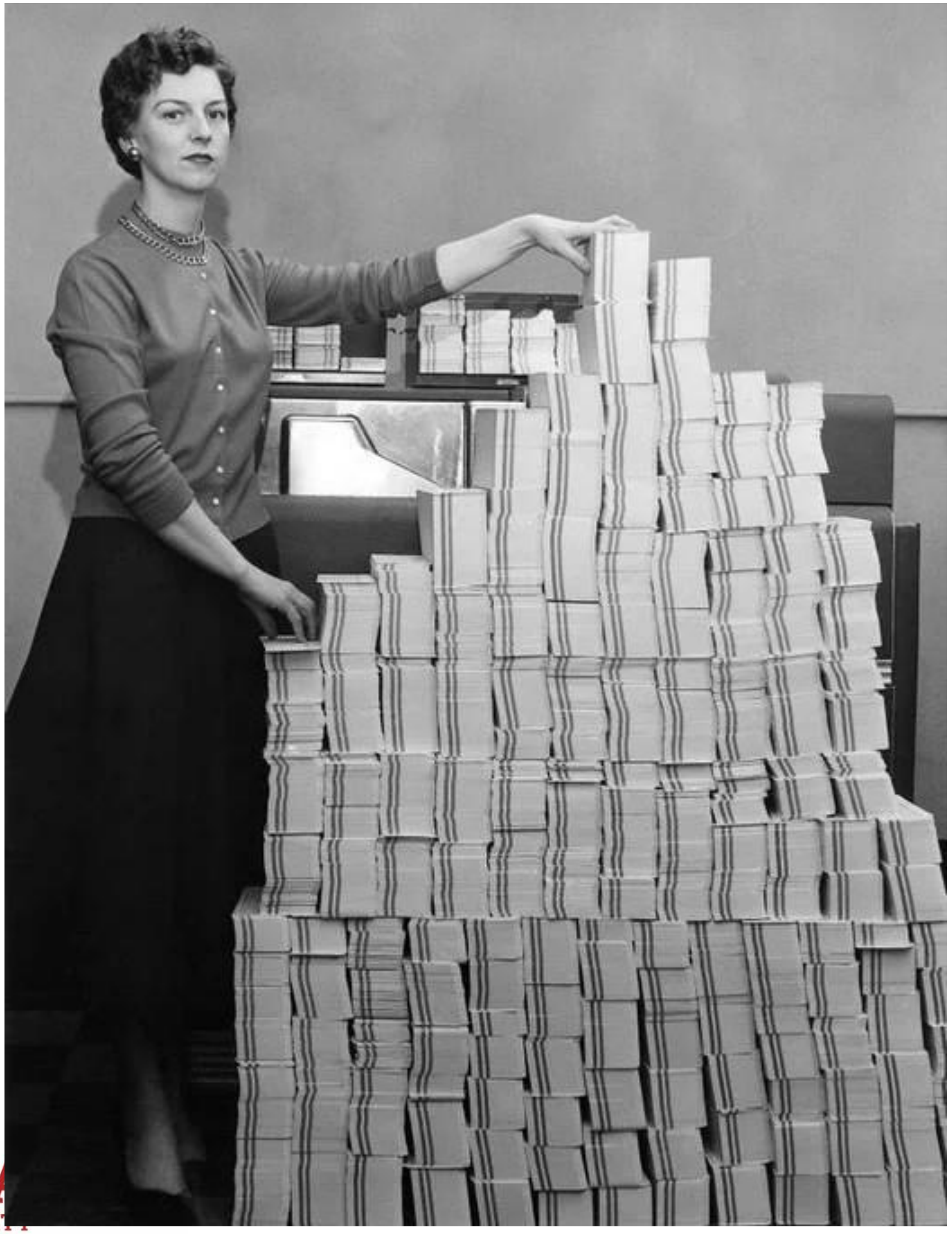


A 12-row/80-column IBM punched card from the mid-twentieth century

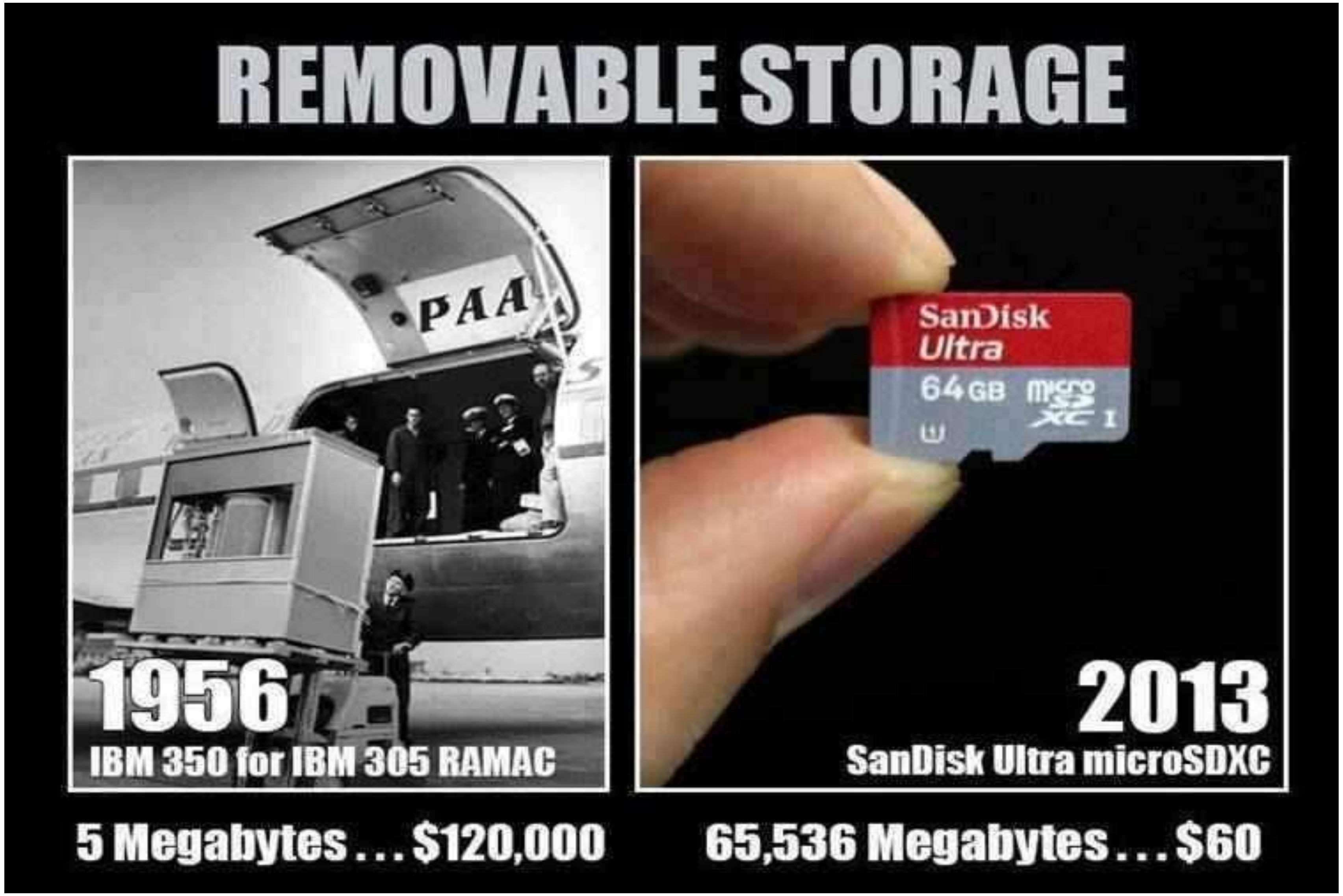
A deck of punched cards comprising a computer program

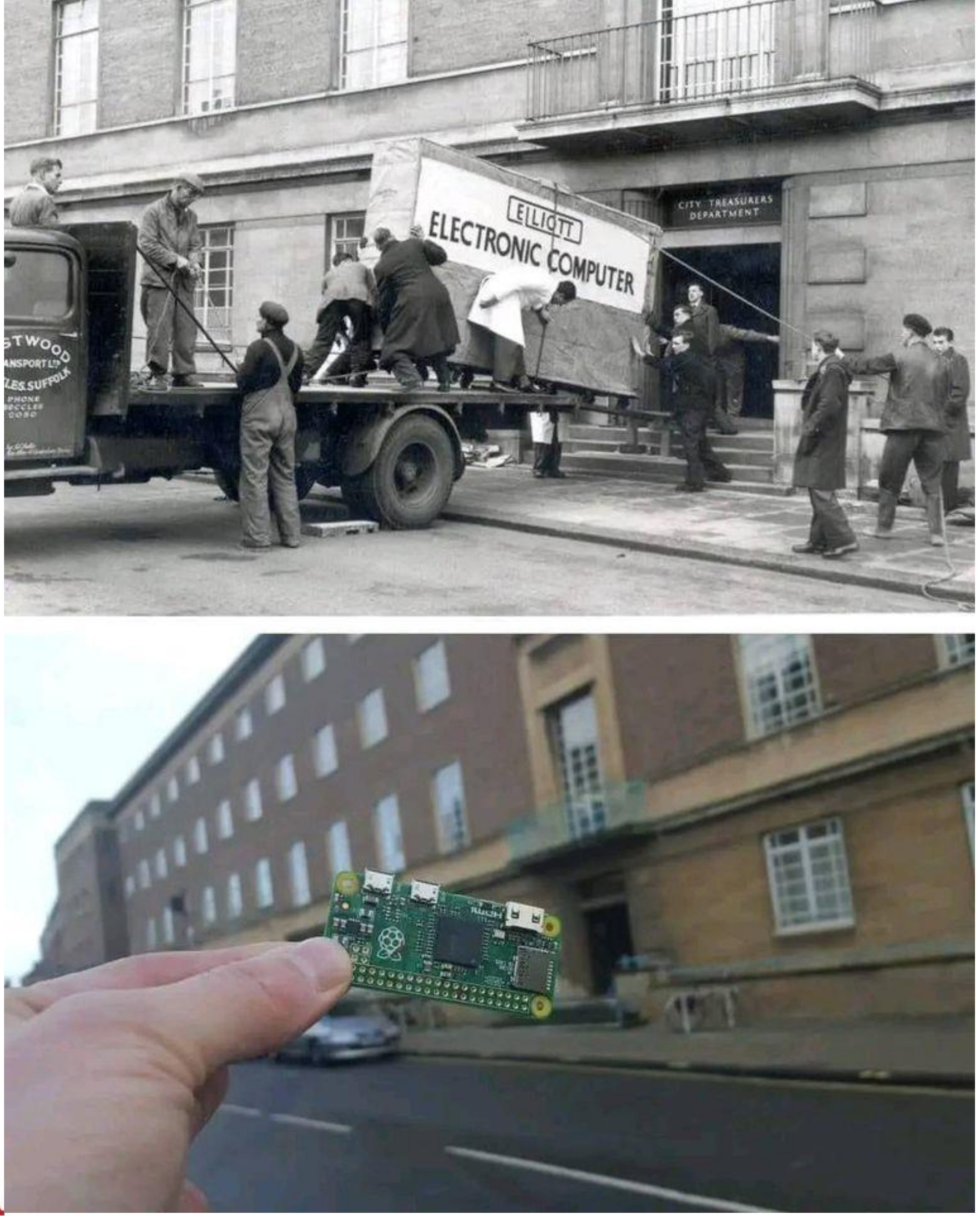
Card reader/punch

Courtesy :https://www.ibiblio.org/comphist/node/57,https://en.wikipedia.org/wiki/Punched_card



- 5 MB of data on 62,500 punch cards in 1955.


REMOVABLE STORAGE
PAA
1956
IBM 350 for IBM 305 RAMAC
SanDisk
Ultra
64GB
micro SD XC I
2013
SanDisk Ultra microSDXC
5 Megabytes . . . $120,000
65,536 Megabytes . . . $60

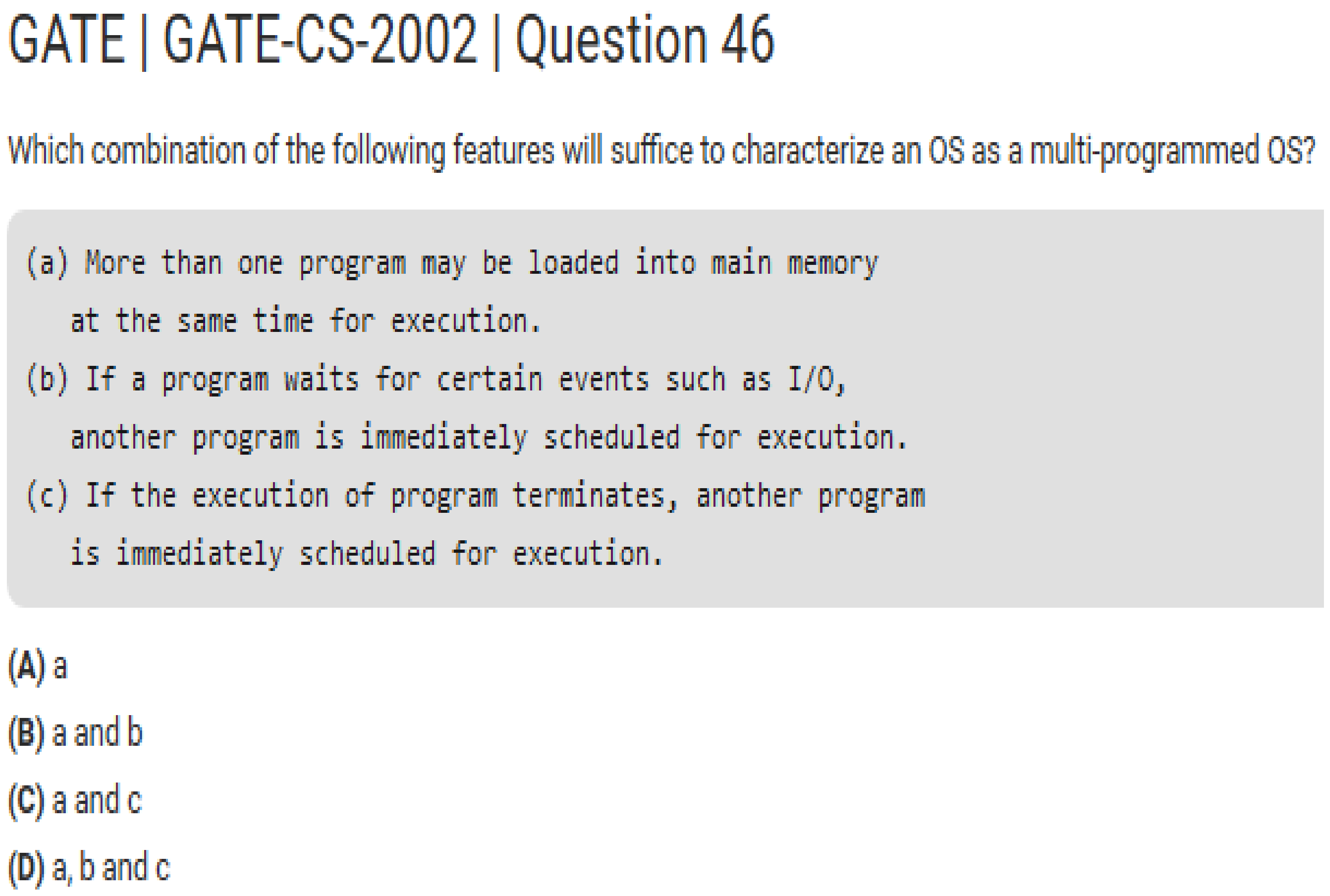


- Same place, same memory, 58 years later

ISRO | ISRO CS 2007 | Question 25

What is the name of the technique in which the operating system of a computer executes several programs concurrently by switching back and forth between them?

(A) Partitioning
(B) Multi-tasking
(C) Windowing
(D) Paging



GATE | GATE-CS-2002 | Question 46

Which combination of the following features will suffice to characterize an OS as a multi-programmed OS?

```
(a) More than one program may be loaded into main memory
   at the same time for execution.
(b) If a program waits for certain events such as I/O,
   another program is immediately scheduled for execution.
(c) If the execution of program terminates, another program
   is immediately scheduled for execution.
```

(A) a

(B) a and b

(C) a and c

(D) a, b and c

Question ?