Components of OS

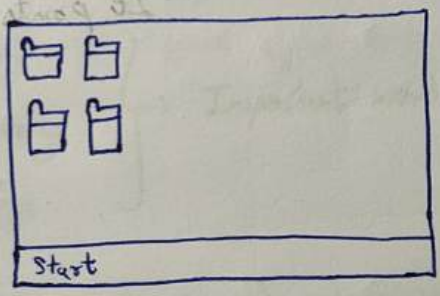
1. User space (Apps are run here)
2. Kernel (has access to underlying hardware)

(1) User space

→ No hardware access

→ Convenient environment for user apps. Provides

When we ON computer windows → ON →



→ GUI

{ Apps don't have privileged access to the underlying hardware. It interacts with kernel. }

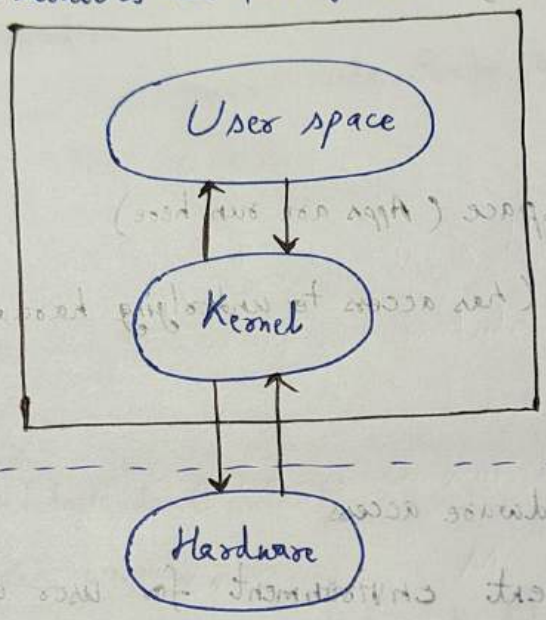
- **GUI**
  - Graphical user interface
  - eg:- Simply right click
    - new
      - folder
- **CLI**
  - Command line interface
    - → Terminal
    - → Power shell
  - eg:- mkdir folder7

→ Shell is that part of OS that receives commands from the users and gets them executed

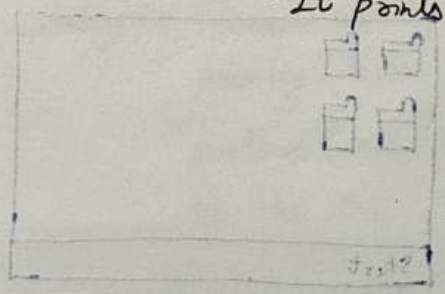
→ Interacts with kernel.
→ In user space application software runs.

② Kernel
↳ Heart of OS

→ Interacts with hardware
→ A kernel is that part of the OS which directly interacts with the hardware and performs the most crucial task.



eg :- HelloWorld.sh
↳ It prints hello world (shell script)



./HelloWorld.sh ← To run

so CLI is user space
↳ so user space tells Kernel to access hardware as user needs some computation. to be done
↳ Kernel requests CPU for computational cycles
↳ CPU executes and displays output.

Functions of Kernel

① Process management

- Process creation, termination
- Process & thread schedule
- Process synchronize
- Process communication

② Memory management

- Allocate/Deallocate.
- Free space management

③ File management

- Create/delete files.
- directory management

④ I/O management

- Management and ~~controlling~~ controlling of all I/O devices

  => Spooling
  => Buffering
  => Caching

  } → Important work

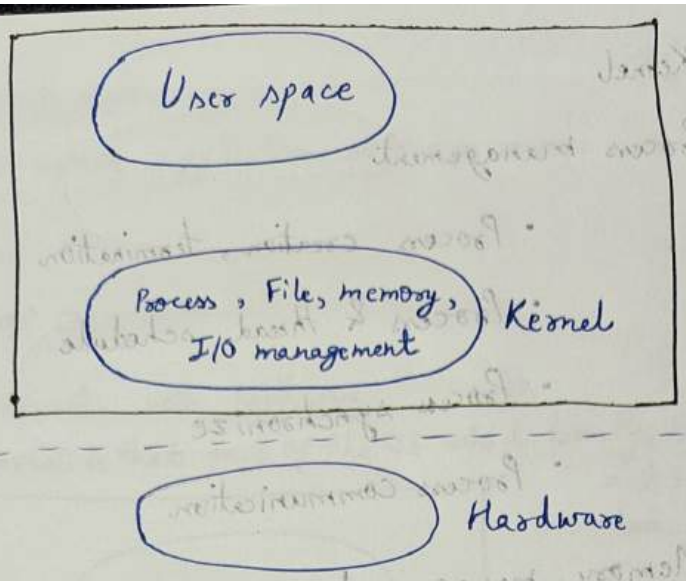
Types of Kernel

① Monolithic Kernel

→ Oldest type of kernel

→ All the OS services run in the same address space as the kernel itself.

→ All the modules are present in kernel so communication between them is efficient.



→ Bulky in size. (kernel)
→ Memory required to run is high
→ Less reliable as if one module crashes the whole kernel is down.
→ High performance as communication is fast.

eg :- Linux, Unix, MS-DOS

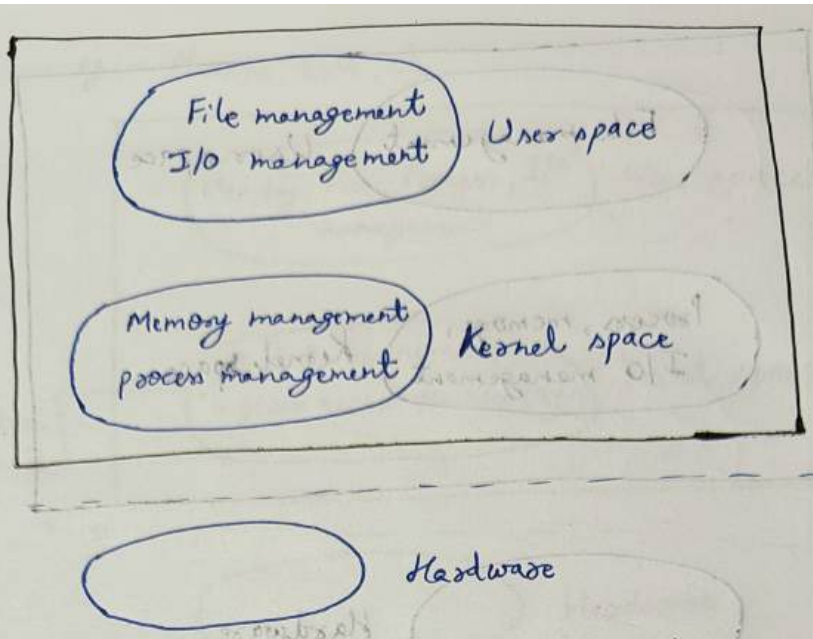
(2) Micro Kernel

→ Only major or core functionalities are in kernel.

eg :- Memory management
Process management

→ Smaller in size (kernel).
→ More reliable (as if a module crashes kernel as still work {if file management crashes kernel can still work})
→ More stable.
→ Performance is slow
→ Overhead switching between user mode and kernel mode.

eg :- L4 Linux, MINIX



eg:- Process first is in user space then it switched to kernel mode as we need to allocate process memory and resources.

↓

then when process is ongoing so it need some I/O operation so it goes to user mode from kernel mode.

↓

then process again wanted memory allocation / deallocation after I/O operation.

⇓

So overhead switching between user mode and kernel mode

↑

leads to slow performance

(3) Hybrid kernel

→ Combined approach

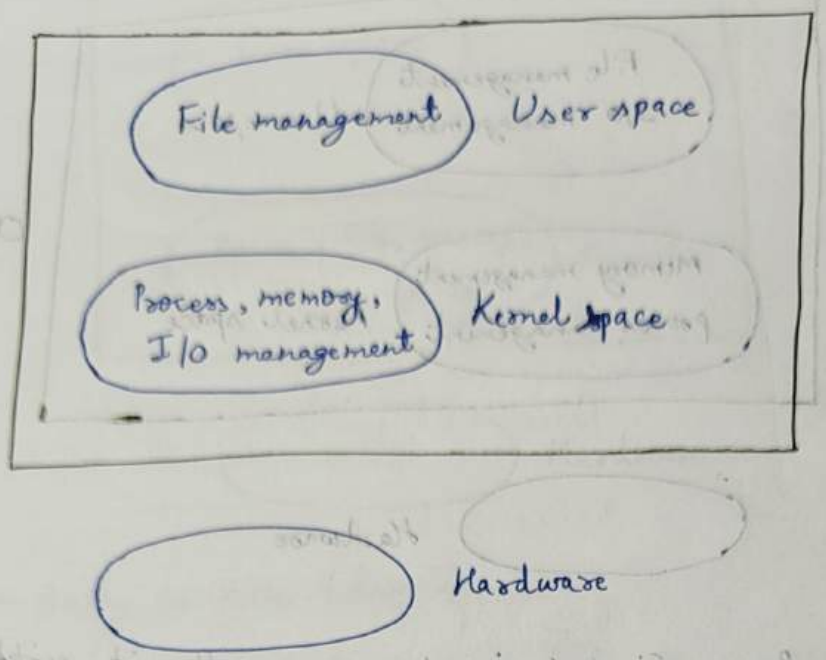
→ Advantages of both worlds.

→ Speed and design of monolithic

→ Modularity and stability of micro

eg:- Mac OS, Windows NT/7/10

→ IPC also happens but lesser overheads.



Extra

(4) Nano/Exo Kernel

- Exo kernel

→ An exokernel is also a very small kernel but with a different idea. It tries to give as much hardware access to apps as possible.

→ Instead of doing things for the application it lets the application do it itself.

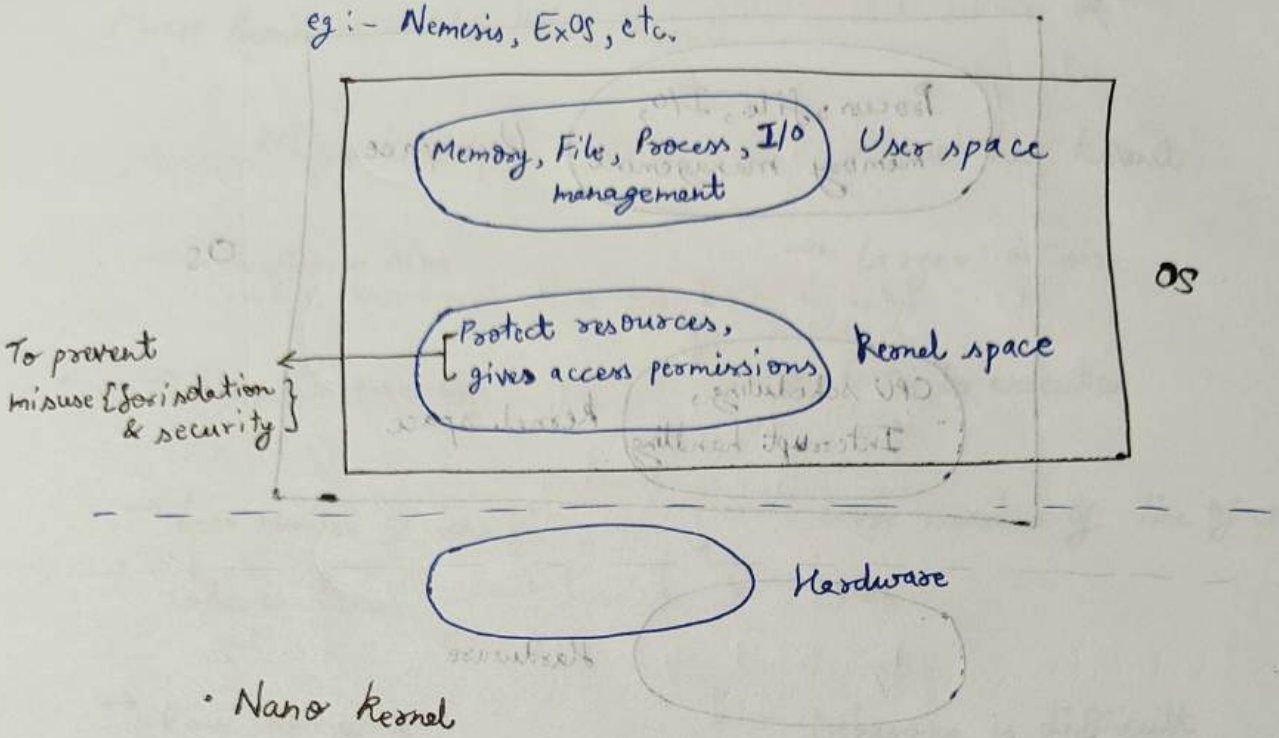
→ Think of it as a security guard who gives you the tools and permissions but lets you build your own house however you want.

→ Gives applications direct control over hardware.

→ Very fast & flexible.

→ Doesn't hide hardware details.

→ Used in research or specialized systems.



- Nano kernel

→ A nanokernel is the smallest possible operating system kernel.

→ It does only the most basic jobs like switching between programs and handling interrupts.

→ It does not manage memory, files or processes directly.

→ Instead it lets other part of OS handle those

→ Think of it as a tiny controller that just keeps everything running at the lowest level.

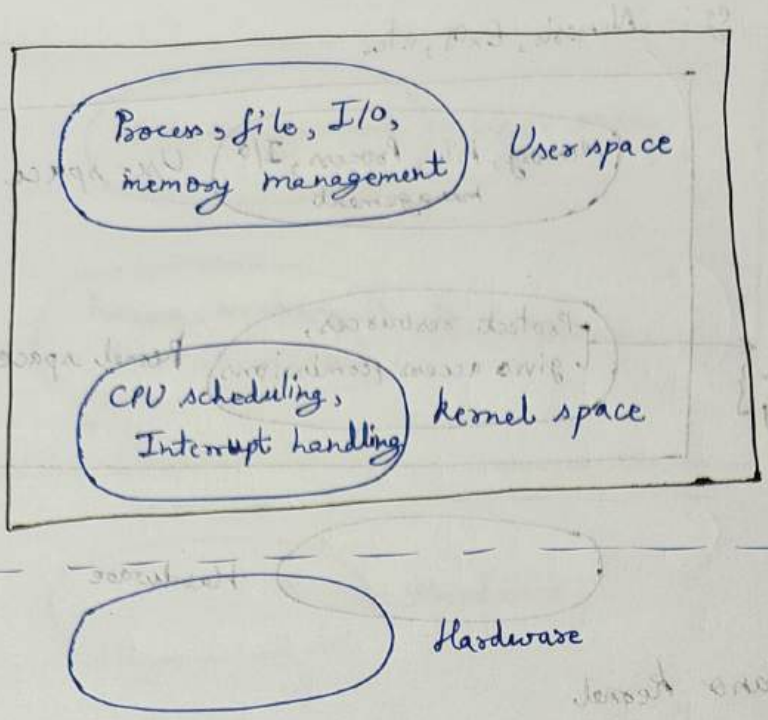
→ Very small size (kernel).

→ Only basic CPU scheduling and interrupt handling.

→ Very fast and secure

→ Used in real-time systems or embedded devices.

eg :- EROS, etc.



Nanokernel vs Exokernel

Feature	Nanokernel	Exokernel
Goal	Be the absolute minimal kernel	Be minimal + give app full hardware access
Who handles user services?	A separate OS layer in user space	Application or libraries directly
Direct hardware access?	Not directly to app	Yes with security.
Control	Kernel just switches context & interrupts	App has control with exokernel protection

Micro kernel vs Monolithic kernel

Micro kernel	Monolithic kernel
→ Smaller in size	→ Larger in size
→ Slower in execution	→ Faster in execution
→ Less number of line of code in kernel	→ Large number of line of code in kernel
→ Easy debugging & management	→ Debugging is difficult (as more lines of code)
→ Easy to add new functionality	→ It is difficult to add new functionality
→ If one component crashes others keep running	→ If one component crashes entire system crashes