

How the applications use the OS services provided?

System Calls: Interface provided by the OS.

ex/1 Make a copy of a file

1. Acquire the file name

- Involves writing to the display (syscall)
- Read input from the keyboard (syscall)

2. Open the source file

- Also open destination file (syscall)

3. Loop reading from source and writing to the destination (syscall)


4. Close the files (syscall)

* API → Application Programming Interface

Java API

WIN API

POSIX API

• A wrapper for system calls 

→ Benefits: Portable

Simple / safe

Elegant

Version control

- API provides a set of functions (Parameters + return values) that are available to the programmer.

- System Calls are involved within the API

• Why?

Portability

Simple

Types of system calls

- [1] • Process management

Execute } kill signal
Signal } (exit) kill -9 terminate
fork (To break)

- [2] File Manipulation

open	delete	read	flush
close	create	write	

- [3] Device Manipulation

read	power	unmount
write	mount	

- [4] Information maintenance

get	set	time / date
-----	-----	-------------

All of these
have some
form of
overlap

5] Communication

send, receive, read, write, socket

6] Protection

(CHMOD, user/group Ids, ...)

7] Memory Management

• Malloc, mmap & munmap etc.

How to Structure the OS?

Large + Complex

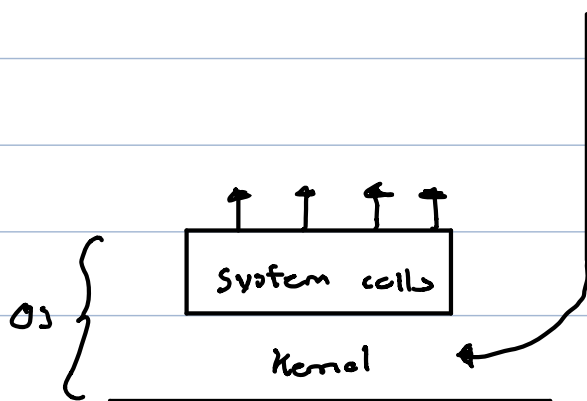
1] Simple

started small but grew beyond original scope

ex // MS DOS

↳ Dish Operating System

Monolithic Kernel



↳ Acts as a whole system

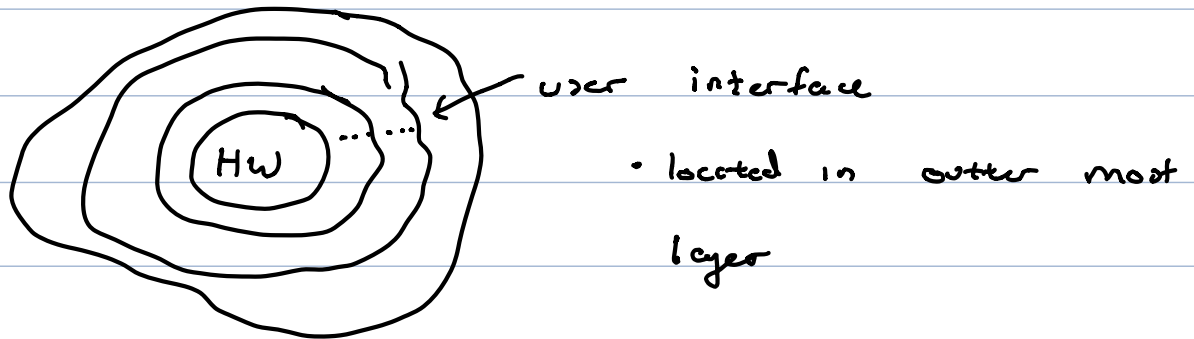
Operates as a whole system

→ When one thing goes wrong
everything goes.

Hardware

2 Layered Approach

- Modularize system design
 - > broken into layers

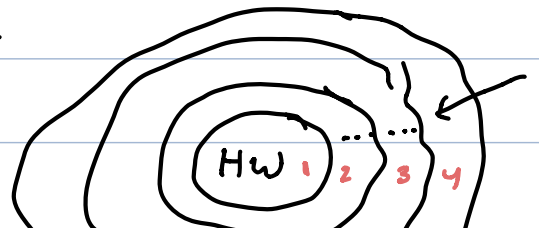


Adv

- Easier to build
- Easier to debug
- Every layer contains abstraction (interface)
- Hiding information
 - You can determine which layer had failed
- The ability to debug each layer independently

Disadv

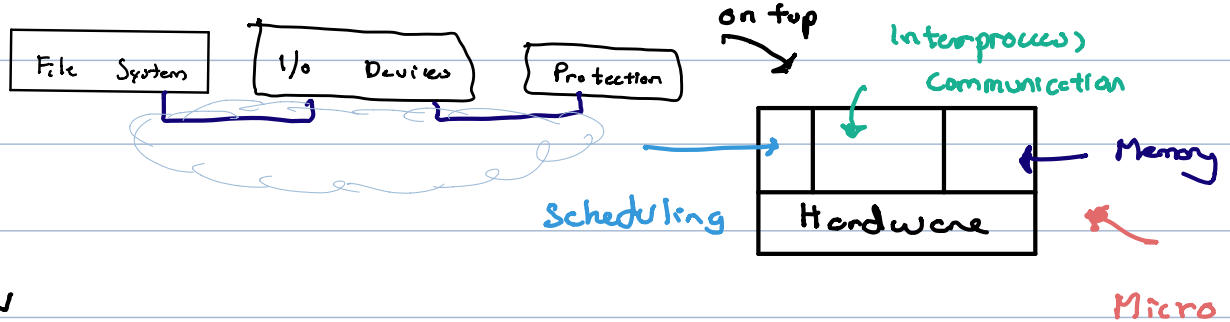
- Dependencies are not clear cut
- How to assign functionality to a particular layer
- Define how many layers
- Each layer depends on the inner layer
- Less efficient



3

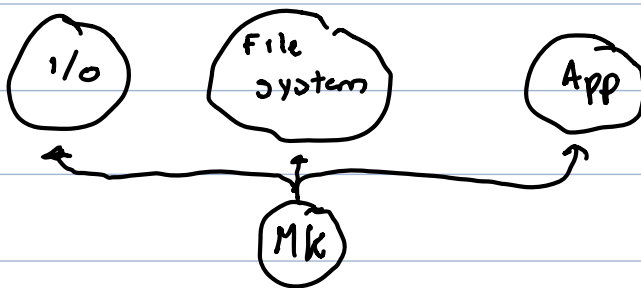
Micro kernels

Idea: Small kernel that is well debugged



Adv

- Debugging
- More fault tolerance
- Easy to extend
- A failing service does not crash the kernel



• Think of it being component based

Interprocess Communication

The middle man

• important method of communicating between services.

• Exchange messages between different services and applications.

Disadv

- A lot of message
- High overhead due to message passing.

4

Modules

Kernel has a set of core components and additional services are added at runtime or boot time, called dynamic loadable modules

