Chapter-I and II
Introduction to OS

Abhijit A. M.
abhijit.comp@coep.ac.in

OS or kernel?

- Debate on terminology
- What you use in daily life is not kernel, but
  - GUI, Shell, Applications, …
  - One view: OS = kernel + (GUI, Shell, Libraries, System programs, Minimum Applications,…)
  - Correct, IMO!
- What we study in this course is kernel
  - Not how is “Ubuntu” built!

Building an “OS”

- E.g. Debian
  - A collection of thousands of applications, libraries, system programs, … and Linux kernel !
  - Linux kernel is at the heart, but heart is not a human without the body !
  - Job of “Debian Developers”
    - Collect the source code of all things you want
    - Create an “Environment” for compiling things : bootstrap challenge
    - Compile everything
    - Ensure that things work “with each other”
      - Very difficult! Versions, dependencies !
    - “Package” things (e.g. make .deb files)

Components of a computer system
Abstract view

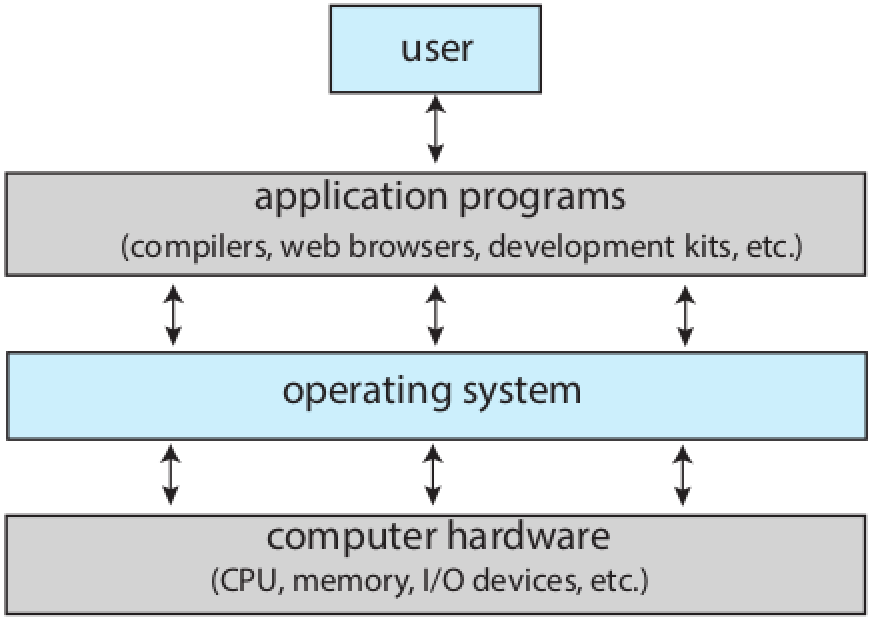
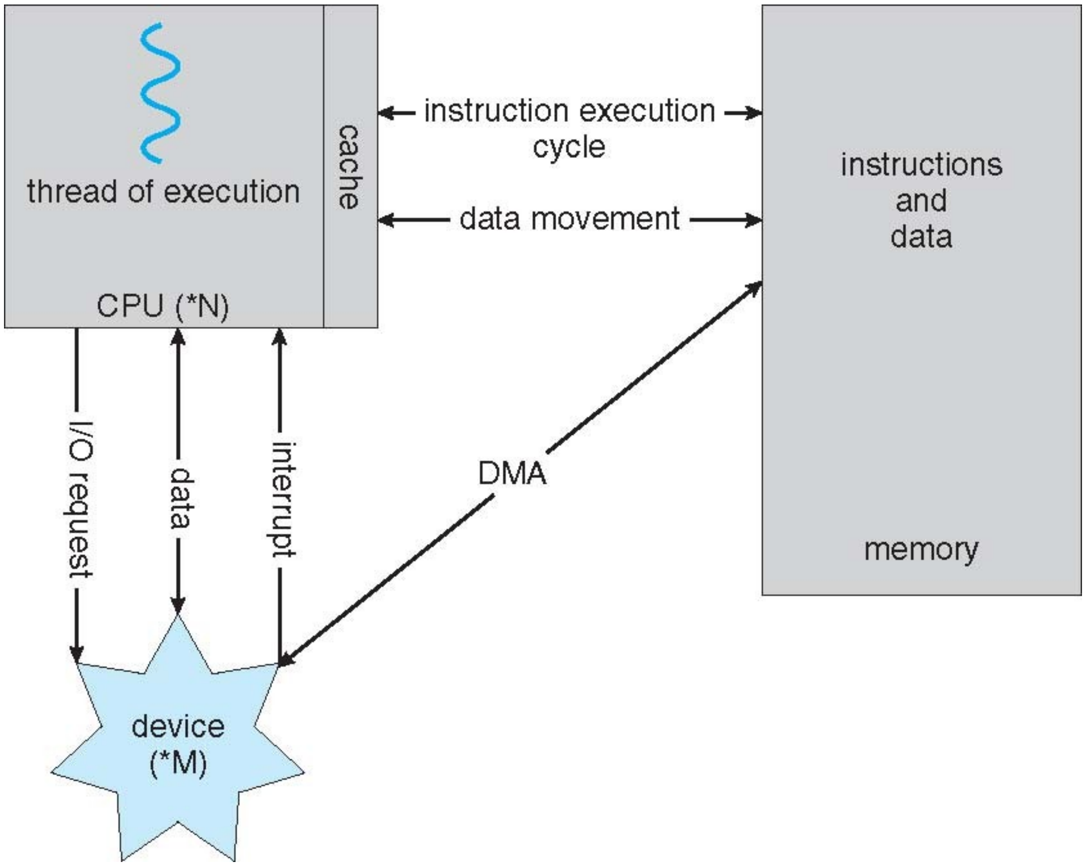


Figure 1.1 Abstract view of the components of a computer system.

Today, different type of kernels to serve different needs

- Each "Computing Environment" has a different requirement of managing resources, processes, etc
  - Desktop / Laptop
  - Thin Clients
  - Workstations
  - Handheld
  - "Servers"
  - Distributed systems
  - Peer to peer systems
  - Virtualization
  - Cloud
  - Real time

Von Neumann architecture

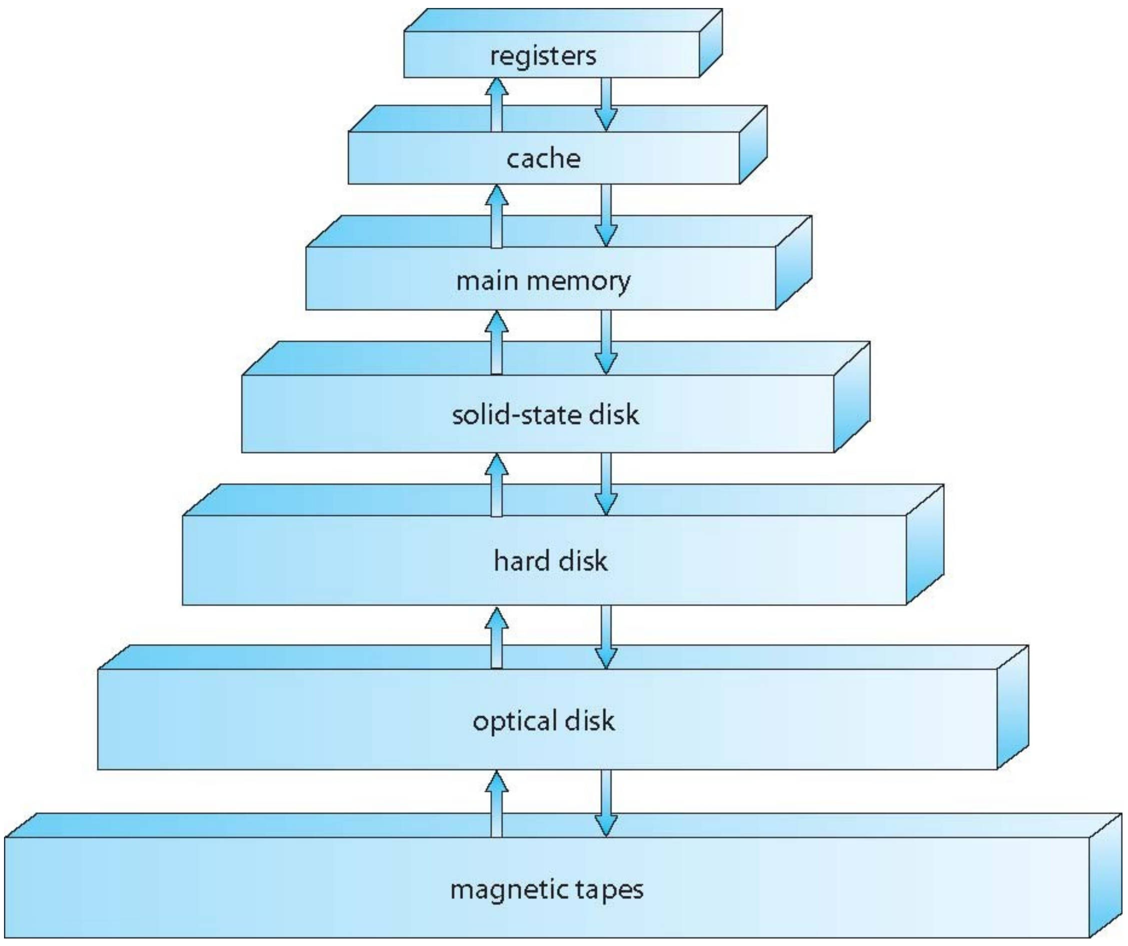


A von Neumann architecture

A key to “understanding”

- Everything happens on processor
- Processor is always running some instruction
- We should be able to tell possible execution sequences on processor

Memory hierarchy



Memory hierarchy

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

System calls

- **We have seen**
  - **Fork(), exec(): system calls to create processes**
  - **Open(), read(), write() : system calls to play with files**
- **System call: A service given by kernel**

System calls: the problem

- On a time shared system
  - Applications and kernel run on the CPU taking turns
  - Kernel allocates time to application , by setting a value in timer register, and then makes application run
    - Timer ticks down every clock cycle
  - *Timer hardware interrupt* occurs when timer = 0 , that is time of application is over
  - Kernel runs again

System calls: the problem

- Kernel must provide "services" to applications
  - Create processes
  - Grant access to resources to processes
  - Etc
- Process should be able to "call" kernel's service
- But everything runs on processor!
  - That is both processes and kernel
- Kernel should be able to run particular instructions which processes should not be able to !

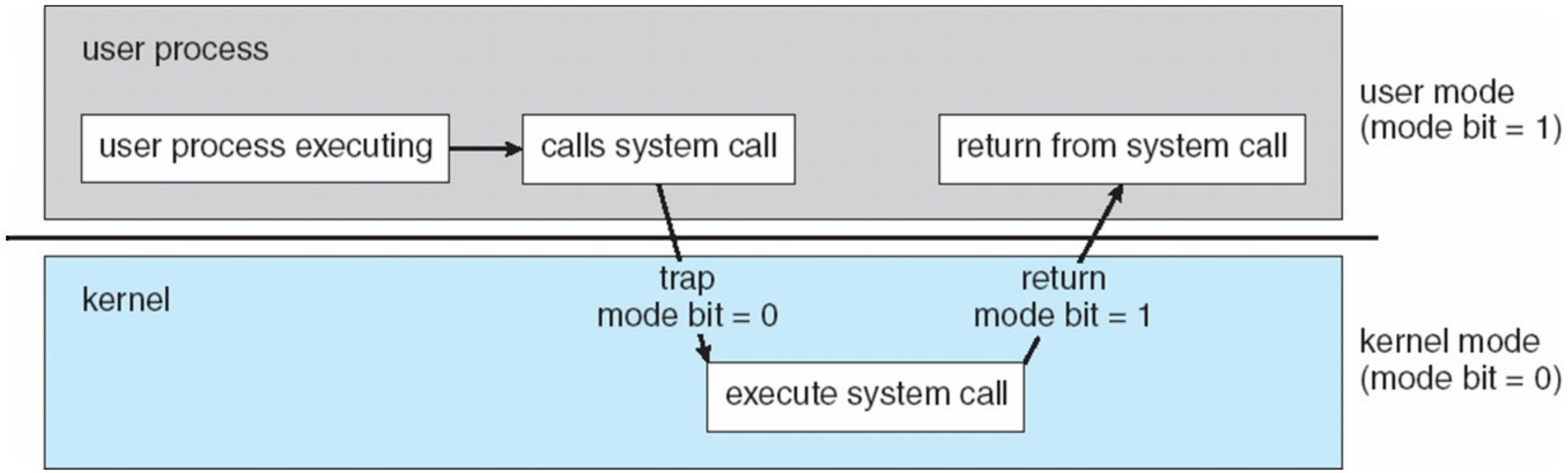
System calls: the problem

- **Privileged Instructions**
  - **Set the timer register**
  - **Set a value in registers of I/O devices**
  - **Set the MMU**
  - **Etc**
- **How to ensure that only kernel code can run privilged instructions?**
  - **Not possible without hardware support in CPU!**

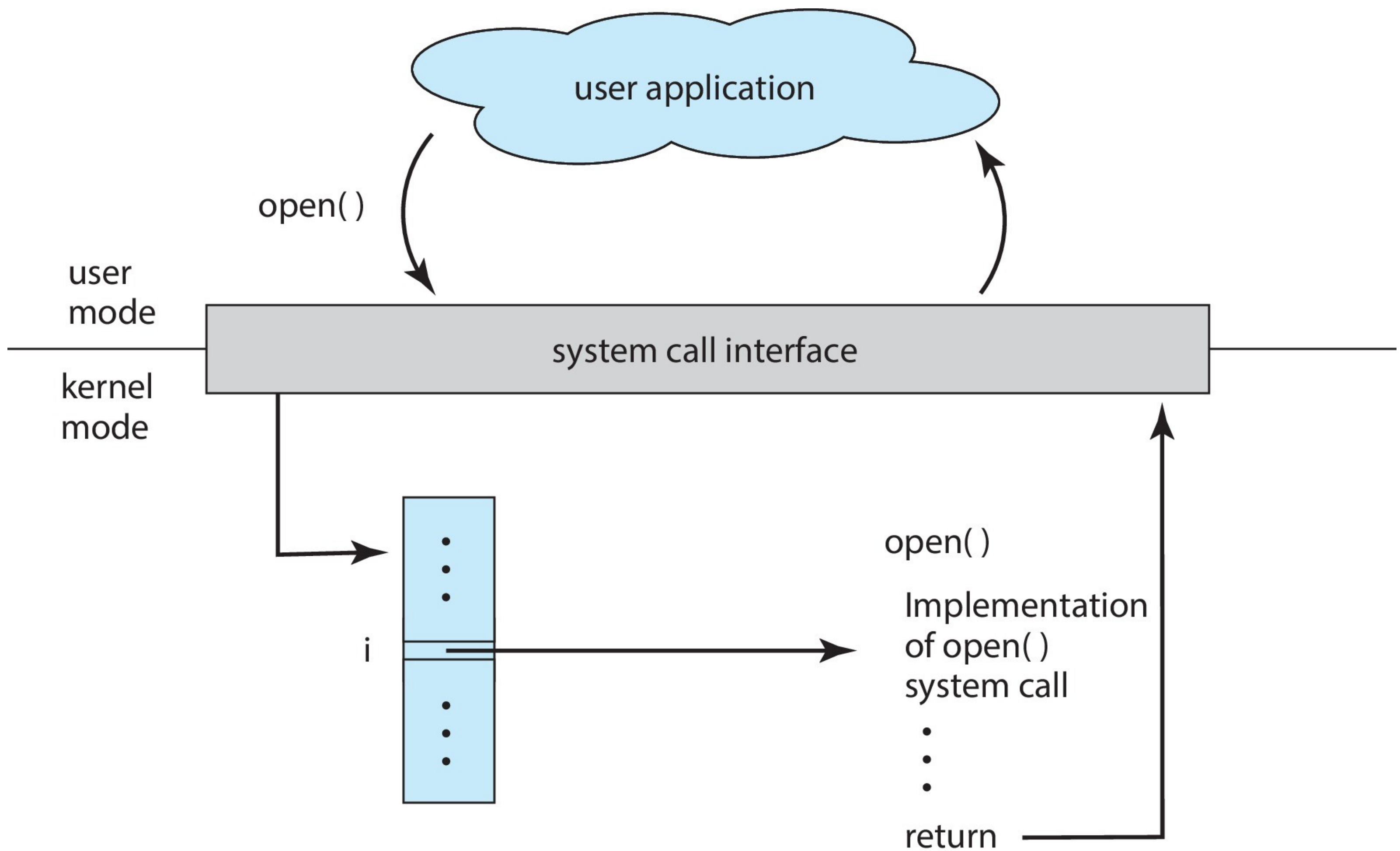
System calls: the problem

- CPU: 2 modes
  - User mode and kernel mode
  - All instructions can be executed in kernel mode
  - Non-privileged instructions can be executed in User mode
  - How to ensure this?
- Special instruction: software interrupt instruction
  - Changes the mode from User Mode to Kernel mode and changes the CS+IP (or PC) to a pre-defined address (where the kernel already exists)
  - Two things at a time – that’s the key

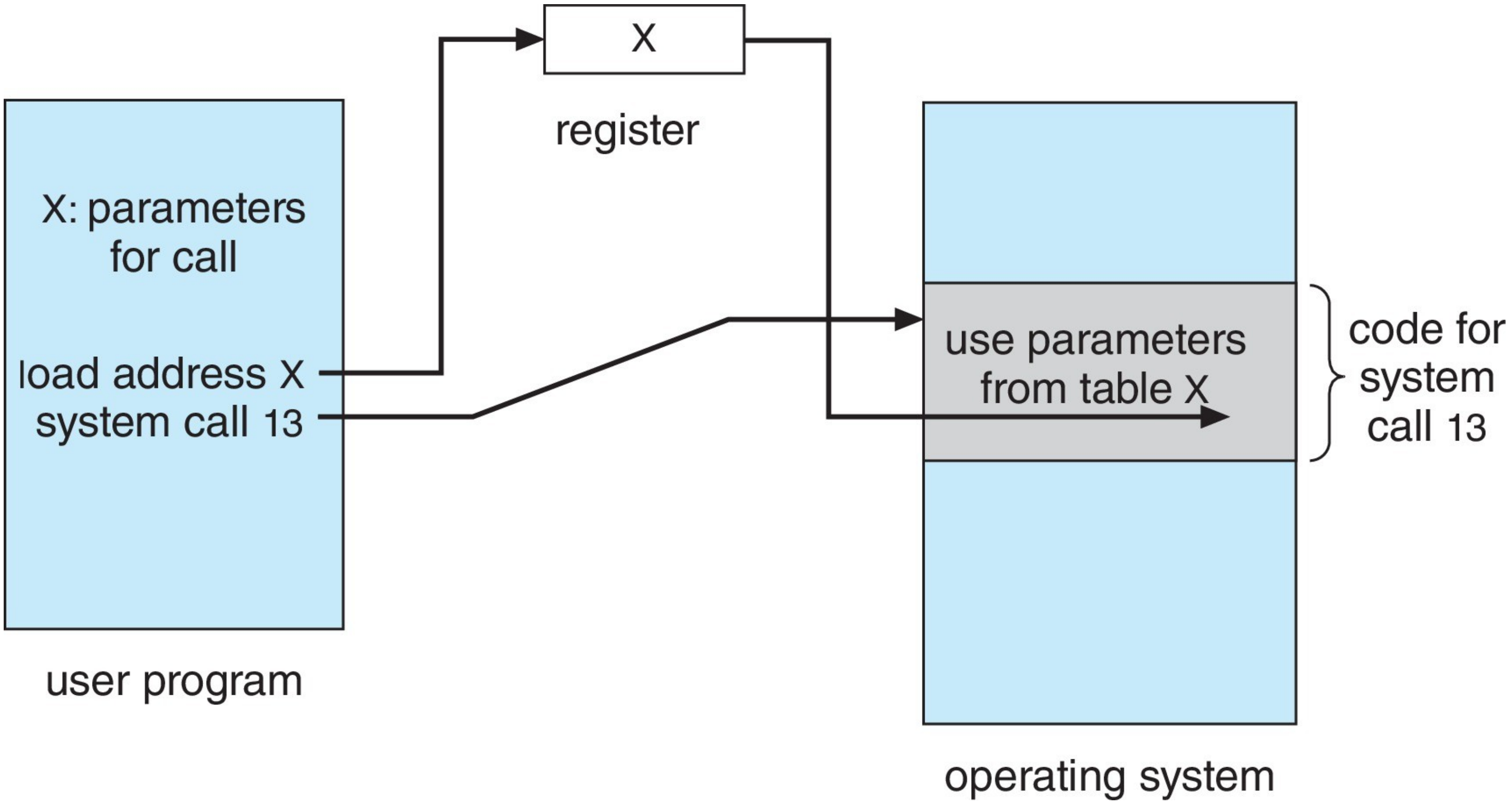
2 modes of operation of CPU needed for multi-tasking system



Application – kernel interaction in system calls



System call – parameter passing



Sharing of data between “user mode” and “kernel mode” code !

What a modern multi-processor, multi-tasking kernel does

- Process Management
  - Create an environment (fork, exec, exit, etc) so that processes can be created, exited, …
- Resource management
  - Processes can access "resources" only through kernel
  - System calls like: open(), brk(), …
  - Includes storage management – enable a "tree" type view of files and folders
  - Manage memory for itself and all processes
  - I/O Management – make I/o devices accessible to processes and itself
  - Manage users (uid, gid, setuid(), seteuid(), ..)
- Protection and Security
  - Limit what a process can do
  - Prevent "attacks" by internal and external agents

System calls exist for …

- Process control
  - create process (fork), terminate process (exit)
  - end, abort (kill)
  - load, execute (exec)
  - get process attributes (getpid, ..), set process attributes (setrlimit, ...)
  - wait for time (sleep, ...)
  - wait event (wait, ...), signal event (signal, kill, ...)
  - allocate and free memory (brk, ...)
  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes

System calls exist for ...

- File management
  - create file (open, create), delete file (unlink)
  - open, close (close()) file
  - read, write, reposition (read, write, lseek, ...)
  - get and set file attributes (chown, chmod, stat, ...)
- Device management
    - request device, release device
    - read, write, reposition
    - get device attributes, set device attributes
    - logically attach or detach devices
  - On Linux: mostly done using file systecalls, but special calls are also there: mknod, mkfifo, ...

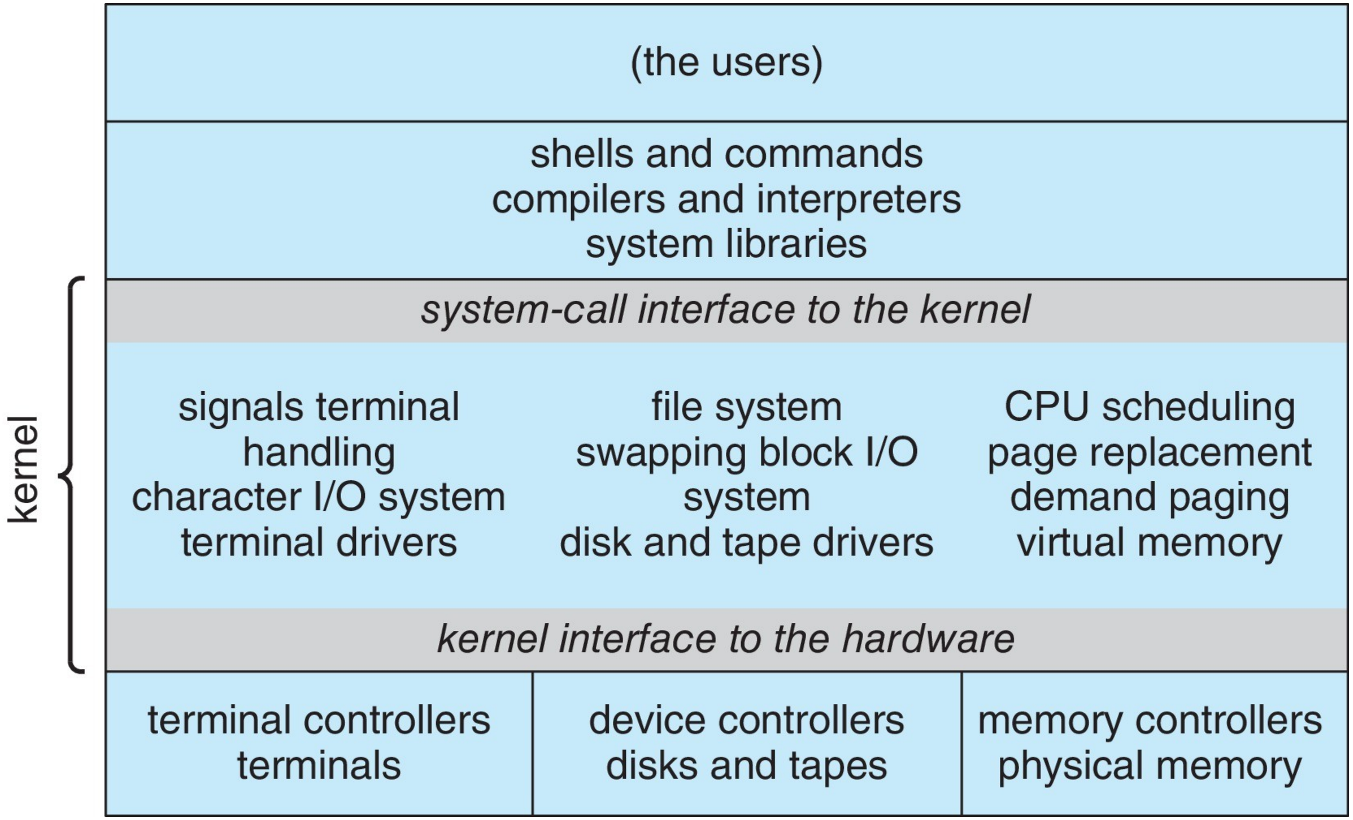
System calls exist for …

- Information maintenance
    - get time or date, set time or date
    - get system data, set system data
    - get and set process, file, or device attributes
  - time, fcntl, …
- Communications
    - create, delete communication connection
    - send, receive messages if message passing model to host name or process name
    - From client to server
    - Shared-memory model create and gain access to memory regions
    - transfer status information
    - attach and detach remote devices
  - Examples: socket, bind, listen, shmget, send, recv,

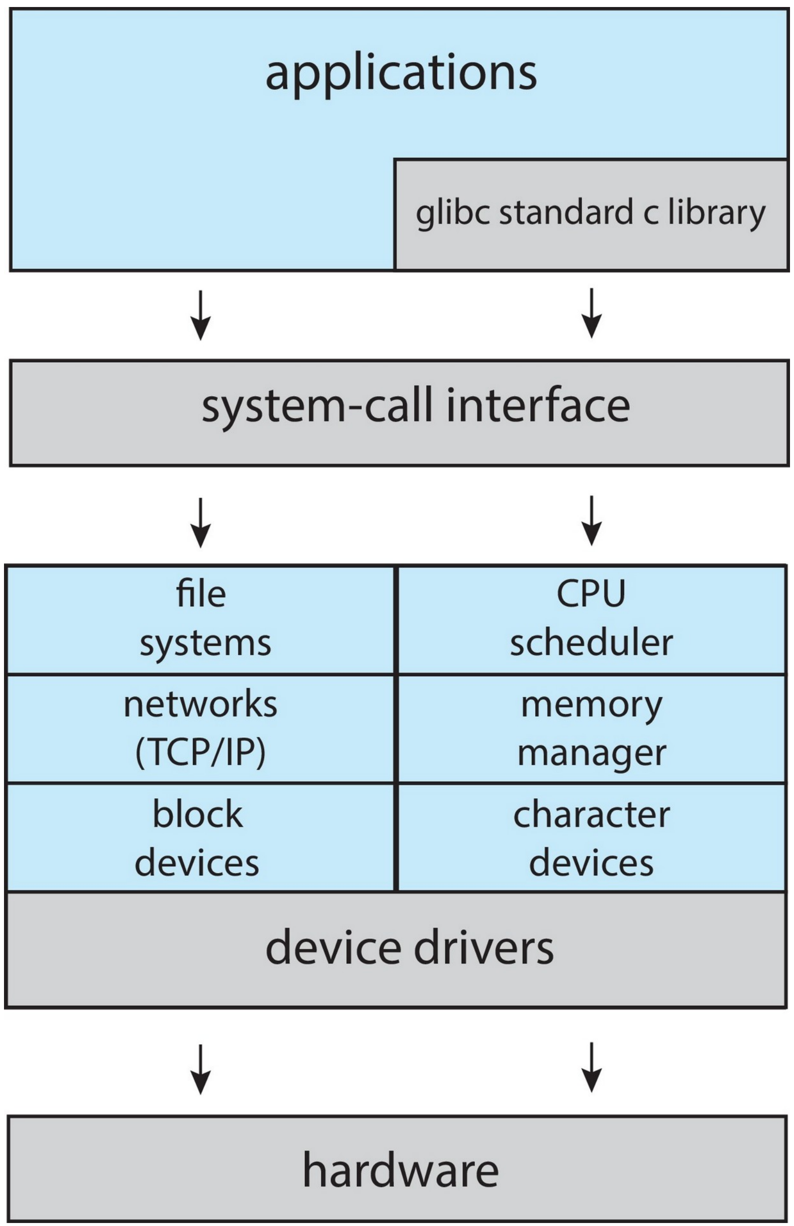
System calls exist for …

- Protection
    - Control access to resources
    - Get and set permissions
    - Allow and deny user access
- More
    - On LinuxL “man syscalls”
    - POSIX: Portable Operating System Interface
    - See output of strace

Traditional Unix system structure



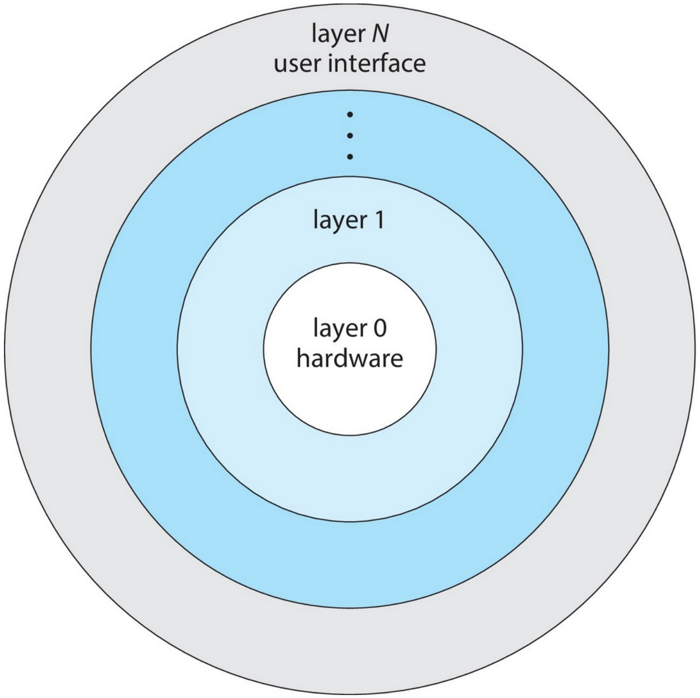
Linux system structure



- **Linux kernel is extendible**
  - **Kernel modules**
  - **lsmod, insmod, ..**

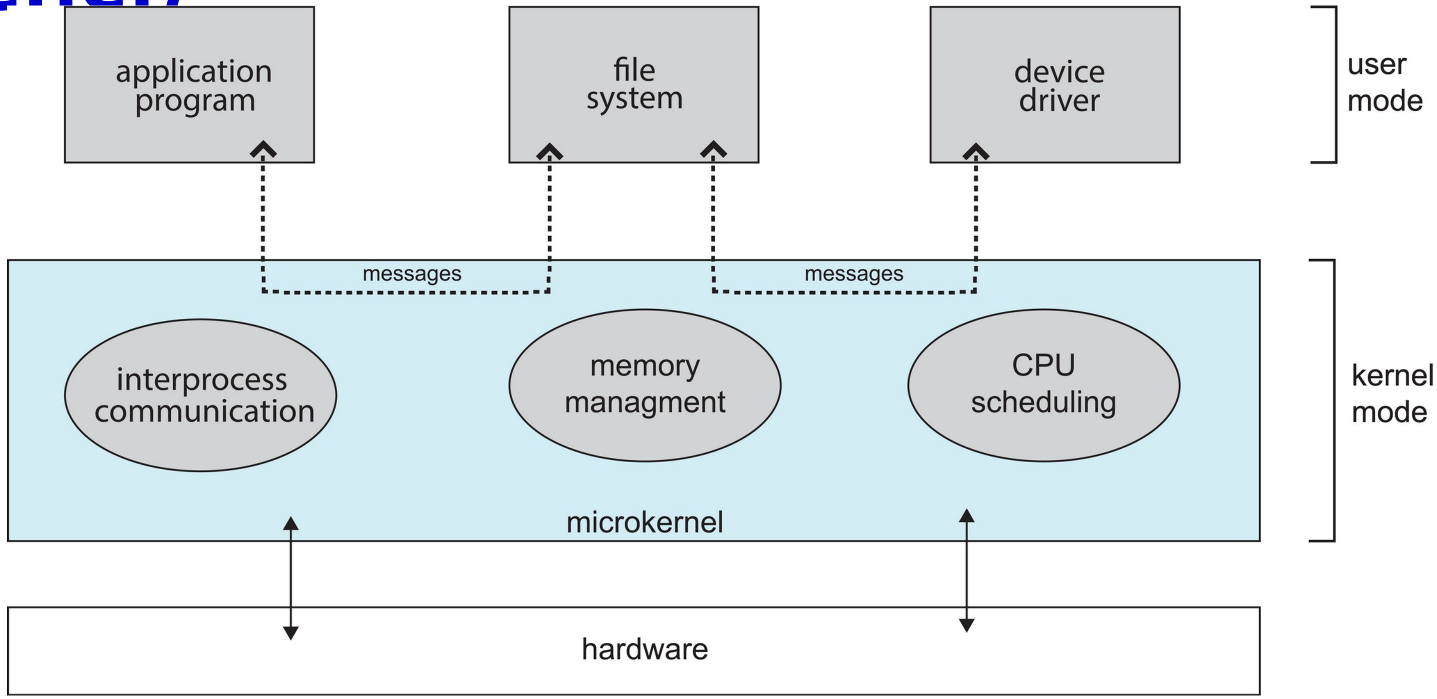
Layered approach

- **xv6**



Microkernel approach

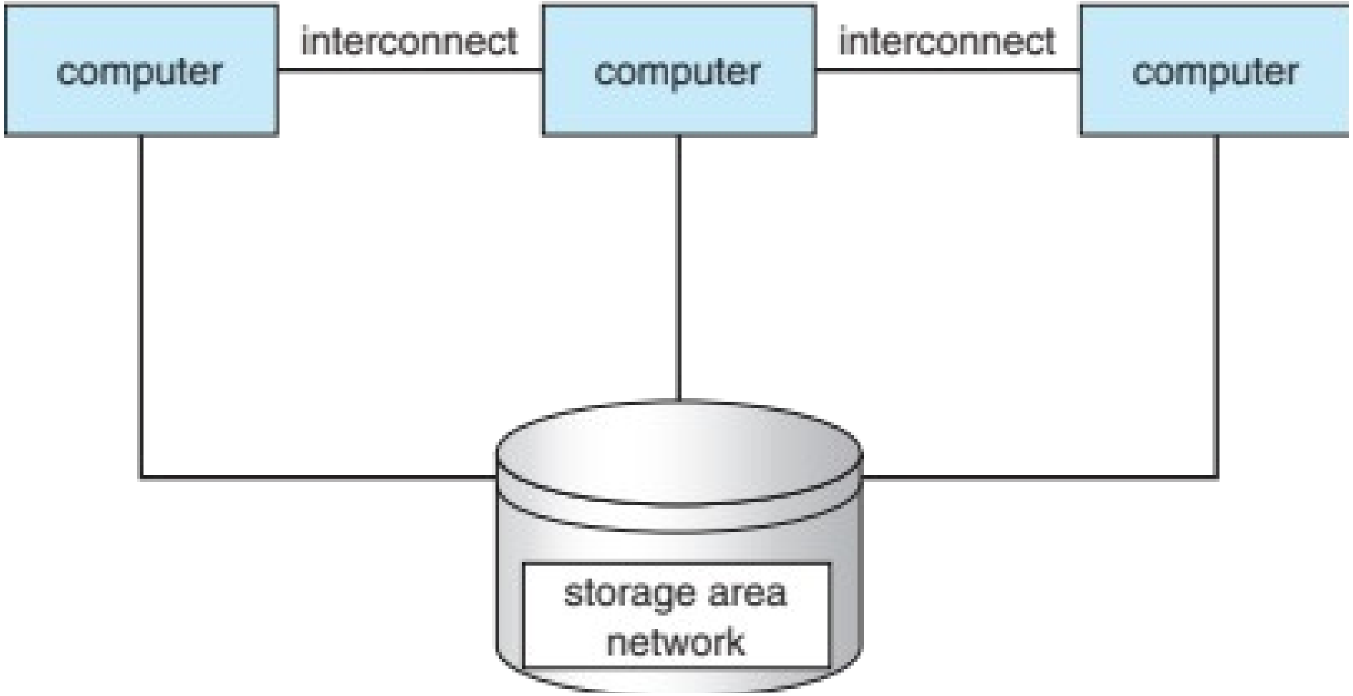
- **Mach, Mac OS X (based on Mach)**
- **GNU herd**
- **Against “monolithic” approach (Linux kerenel)**



Skipped

- IOS, Android structure

Clustered sytem



Free Software / Open Source kernels

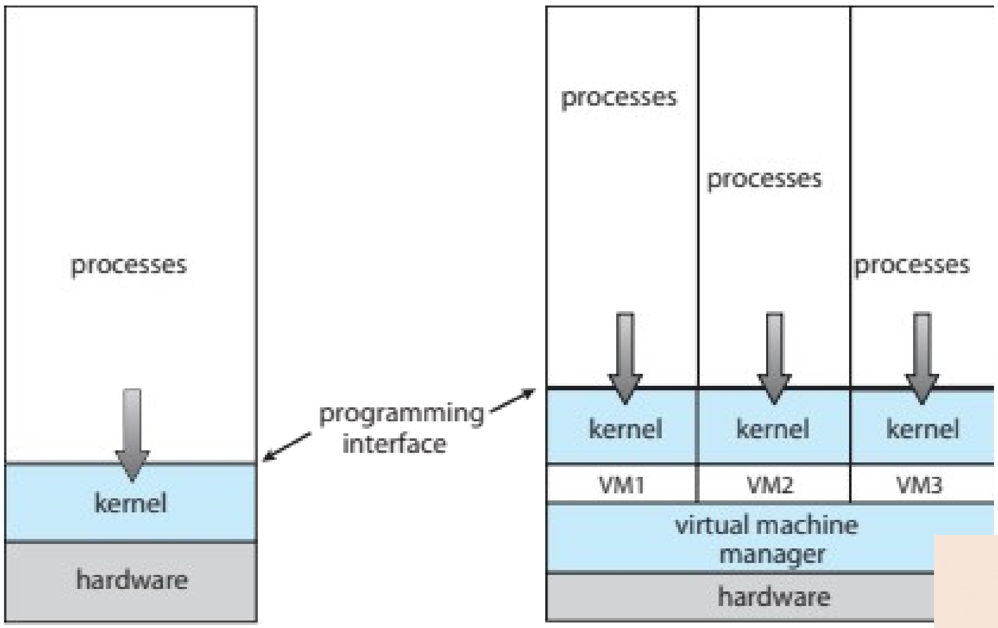
- Linux
- BSD Unix
- GNU Herd
- xv6
- Minix
- FreeRTOS
- OpenSolaris
- Many others

https://en.wikipedia.org/wiki/Comparison_of_open-source_operating_systems

Virtualization

- A general concept!
- Create a “virtual reality” out of “reality”
  - Black boxes that do something
  - End users need not know “how”, but only “what”
- Following all are “virtualizations”
  - Function
  - Class
  - An application
  - Debian GNU/Linux (or any OS)

Virtual Machines, an example



(a) (b)