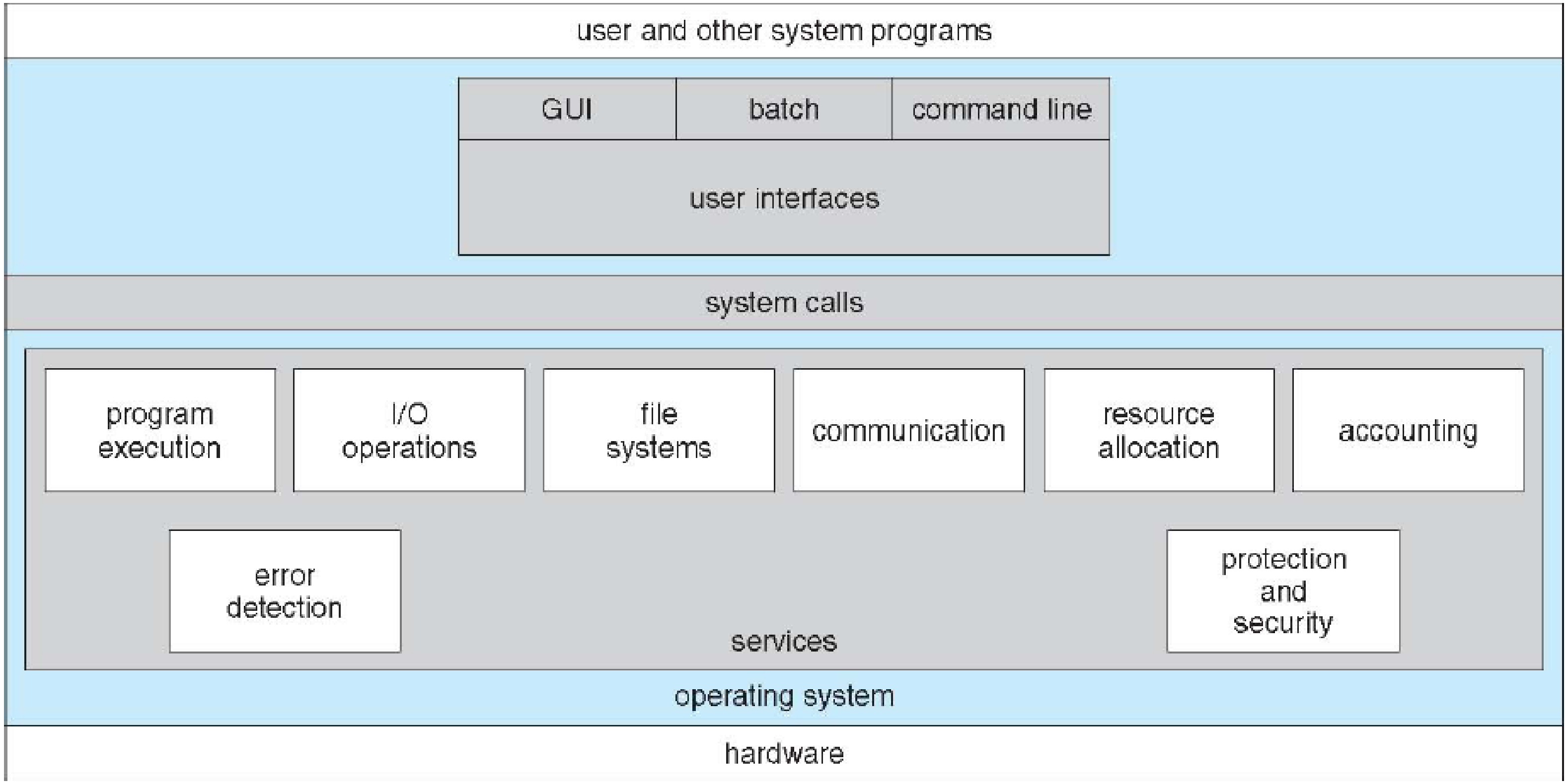
CH 2: Operating System Structure

Operating System Services

- A view of the Operating System services
  - An operating system provides an environment for the execution of programs.
  - It provides certain services to programs and to the users of those programs.



OS Services for User Support

- User Interface
  - CLI: Command-line interface
  - GUI: Graphical user interface
  - batch interface
- Program Execution
  - Load, run and end execution
- I/O operations
- File-system manipulation
  - Read/write/create/delete/search/list files
- Communications
  - shared memory vs. message passing
- Error Detection
  - Errors at the hardware-level or software-level
    - Bit-flip, memory error, power failure, parity, connection failure … etc.

OS Services for Efficient Operation

- Resource Allocation
  - Resource: CPU cycles, memory space, memory bandwidth, file system space, file I/O bandwidth
- Accounting
  - Keeping track of which user (or process) uses how much of the resources
- Protection and Security
  - In multi user environment, all access to system resources must be controlled
  - Other processes should not interfere with other processes

"A system is as strong as its weakest link"

OS Interfaces

- Command Interpreters (Command-line Interface: CLI)
  - Also called as shell: sh, ksh, csh, bash

Example)

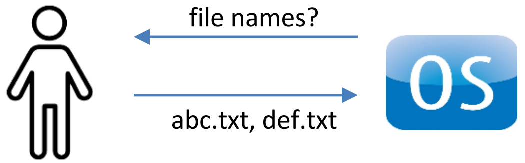
```
$ rm file.txt
```

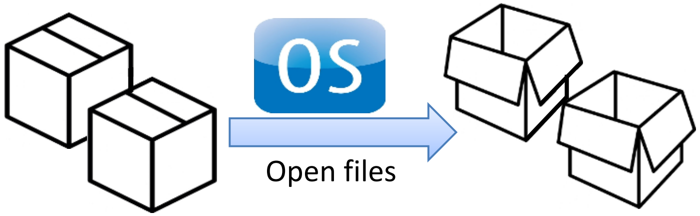
The shell searches for ‘rm’ in the path, load them into memory and execute it with ‘file.txt’ as a parameter

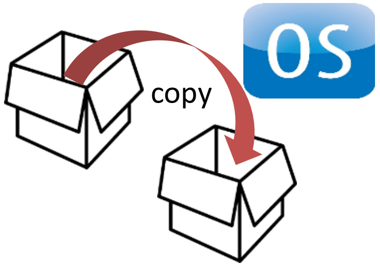
- GUI (Graphical User Interface)
  - mouse, folder, icons, touch screen … etc.

System Calls

- Example of how system calls are used:
  - Read data from one file and copy to another file
    - Request user for two file names
      - prompt use to enter
        - each character being typed is displayed
      - In GUI, icons are shown and user selects the source file
    - Open two files
      - On error, error messages are printed and it terminates
      - Errors: file does not exist, permission error …
    - Copy data and write to the destination file in a loop
      - read and write
    - Close both files
    - Write completion message
    - Terminate the program



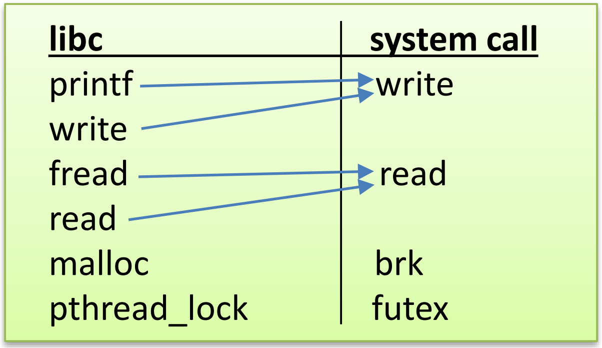




System Calls

- System calls are heavily used <u>*even for a simple task*</u>
  - thousands of system calls per second

- Programmers use system calls *indirectly* through APIs (Application Programming Interface: **set of functions**)
  - Windows API, POSIX API, Java API
  - API on top of another API
    - Java uses libc

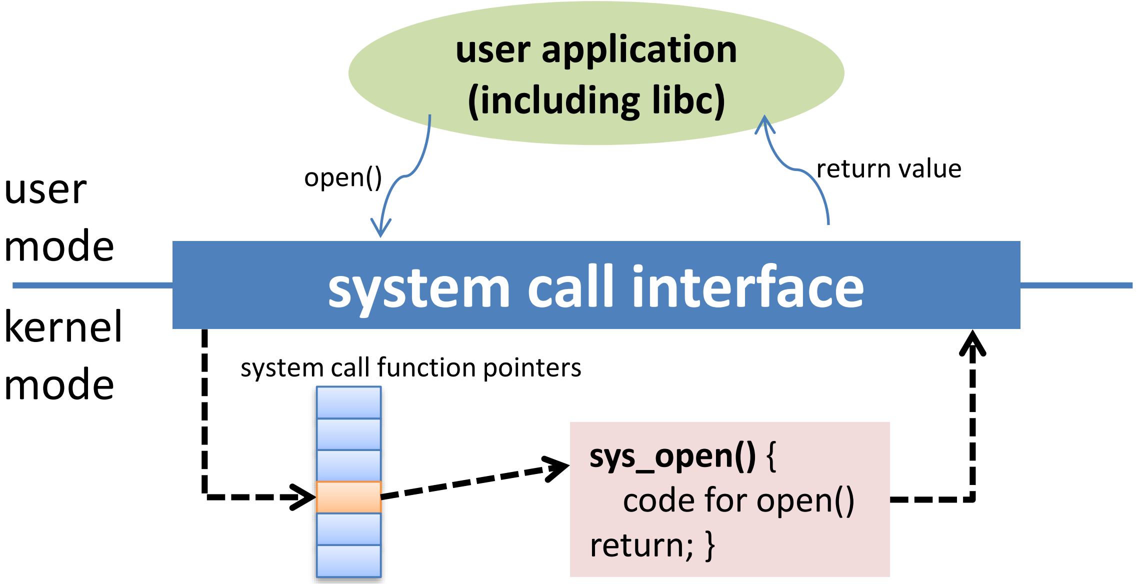


libc	system call
printf	write
write	
fread	read
read	
malloc	brk
pthread_lock	futex

- Why use APIs rather than system calls directly?
  - Directly using system call is difficult because …
  - Portability (What is this?)

System Call Interface

- Set of library functions that links to the system calls



- Caller does not need to know how system call is implemented
- Caller needs to know only the interface and what it returns

NAME

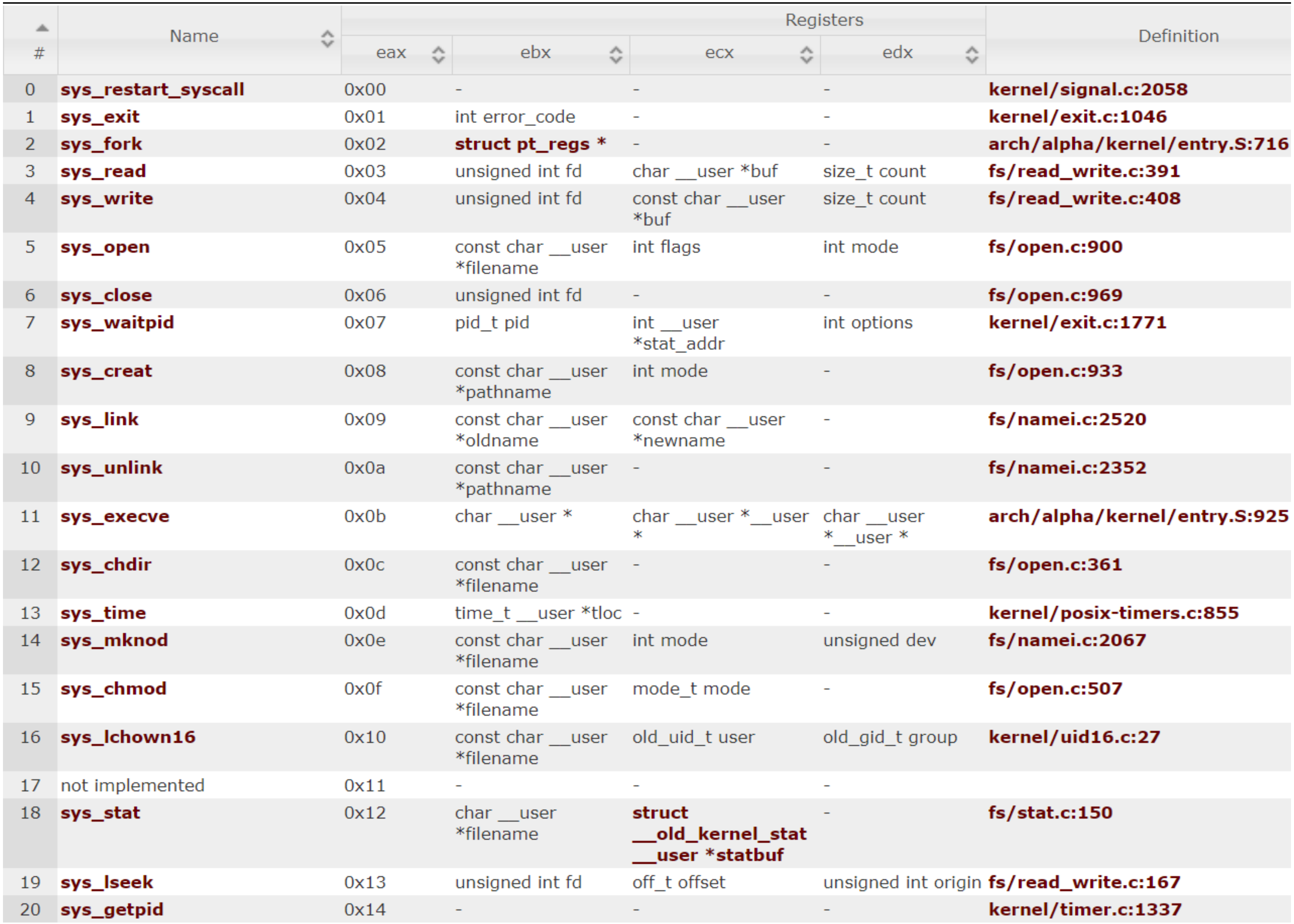
read - read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int fd, **void** *buf, **size_t** count**);**

System Call Interface



#	Name	Registers				Definition
		eax	ebx	ecx	edx	
0	**sys_restart_syscall**	0x00	-	-	-	**kernel/signal.c:2058**
1	**sys_exit**	0x01	int error_code	-	-	**kernel/exit.c:1046**
2	**sys_fork**	0x02	**struct pt_regs ***	-	-	**arch/alpha/kernel/entry.S:716**
3	**sys_read**	0x03	unsigned int fd	char __user *buf	size_t count	**fs/read_write.c:391**
4	**sys_write**	0x04	unsigned int fd	const char __user *buf	size_t count	**fs/read_write.c:408**
5	**sys_open**	0x05	const char __user *filename	int flags	int mode	**fs/open.c:900**
6	**sys_close**	0x06	unsigned int fd	-	-	**fs/open.c:969**
7	**sys_waitpid**	0x07	pid_t pid	int __user *stat_addr	int options	**kernel/exit.c:1771**
8	**sys_creat**	0x08	const char __user *pathname	int mode	-	**fs/open.c:933**
9	**sys_link**	0x09	const char __user *oldname	const char __user *newname	-	**fs/namei.c:2520**
10	**sys_unlink**	0x0a	const char __user *pathname	-	-	**fs/namei.c:2352**
11	**sys_execve**	0x0b	char __user *	char __user *__user *	char __user *__user *	**arch/alpha/kernel/entry.S:925**
12	**sys_chdir**	0x0c	const char __user *filename	-	-	**fs/open.c:361**
13	**sys_time**	0x0d	time_t __user *tloc	-	-	**kernel/posix-timers.c:855**
14	**sys_mknod**	0x0e	const char __user *filename	int mode	unsigned dev	**fs/namei.c:2067**
15	**sys_chmod**	0x0f	const char __user *filename	mode_t mode	-	**fs/open.c:507**
16	**sys_lchown16**	0x10	const char __user *filename	old_uid_t user	old_gid_t group	**kernel/uid16.c:27**
17	not implemented	0x11	-	-	-	
18	**sys_stat**	0x12	char __user *filename	**struct __old_kernel_stat __user *statbuf**	-	**fs/stat.c:150**
19	**sys_lseek**	0x13	unsigned int fd	off_t offset	unsigned int origin	**fs/read_write.c:167**
20	**sys_getpid**	0x14	-	-	-	**kernel/timer.c:1337**

System Call Types

- Six categories
  - Process control
    - **fork, exec, exit …**
  - File manipulation
    - **create, open, close, read, write, lseek**
  - Device manipulation
    - **open, close, read, write, ioctl**
  - Information maintenance
    - **time, date, pid**
  - Communications
    - **open, close, connect, accept, read, write, send, recv, pipe, mmap, sendfile …**
  - Protection
    - **Chmod, umask, chown …**

System call types: Process control

- Type of tasks
  - end, abort
  - load, execute
  - create or terminate process
  - get/set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- Scenarios
  - Error handling
    - Needs to terminate → creating a memory dump → print message → run next command
    - In GUI, pop-up window might be shown
    - In batch system, whole job is aborted
  - Loading and executing another program
    - Where to return the control to when new process terminates
    - Whether to concurrently run
  - In multiprogramming, we need to control processes
    - Get/set process attributes (e.g., priority, time limit)
    - Selectively terminate unneeded processes

System call types: Communications

- Two models of communications
  - *message passing* and **shared-memory** model
- Message passing
  - based on exchanging messages
  - connection must be first opened
    - What does the 'opening connection' really mean?
    - Sender/Receiver identities must be known
- Shared-memory model
  - In normal condition, processes are not allowed to access other processes' memory
  - System call for sharing must be invoked
  - Process must be careful to maintain the integrity of data in the shared memory region

Other types

- File management
  - create/delete file
  - open/close
  - read/write/reposition
  - get/set file attributes
    - file name, type, permissions, time, uid … etc.
- Device management
  - request/release device: for exclusive usage ↔ open/close of files
  - read/write/reposition
  - get/set device attributes
  - logically attach/detach devices
- Information maintenance
  - get/set time or date
  - get/set system data: list of users, version info, free mem, disk usage
  - get/set process, file, device attributes
    - process execution time
    - dump, trace

OS Design and Implementation

- Design goals: Requirements can be divided into two
  - User goals
    - ease of use, reliable, ease of learning, fast
  - System goals
    - ease of implementation, ease of maintenance, ease of operation, flexible, efficient, fault-tolerant

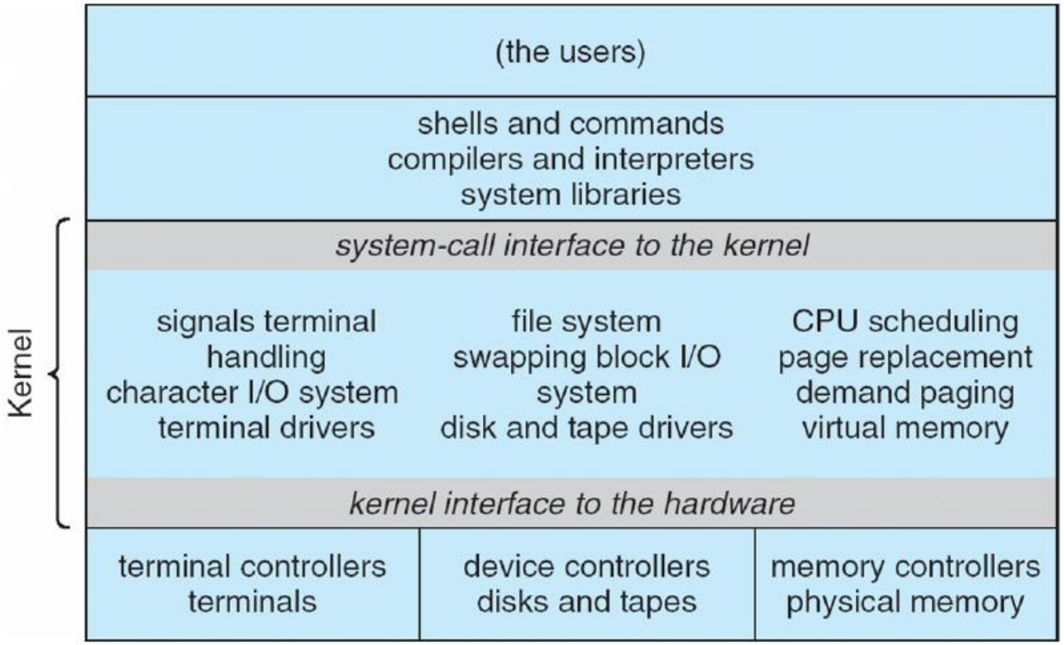
  → **Vague requirements and no clear solution**
- Guideline: Mechanism and Policy
  - Separation of policy from mechanisms
    - mechanisms: how
    - policy: what
  - It is a technique frequently applied to many cases
    - ex) timer mechanisms and scheduling policy
    - ex) password expires in 30 days
  - Why separate them? Policy changes.
    - If policy change requires mechanisms change, it is not desirable.

OS Implementation

- Implementing in assembly language
  - tedious, difficult, error-prone, difficult to port
- Implementing in high-level language
  - Advantages
    - Faster implementation time
    - Compact and easier to understand/debug
    - Improved complier will improve the generated code
    - Easier to port
      - MS-DOS (written in asm) needs emulator
      - Linux is mostly in C, no need to emulate, run natively
  - Disadvantages
    - Performance: potentially sub-optimal code generation
      - Not true in today's compiler technology
      - complier is better at handling complex dependencies
    - Compromise
      - small critical code can be first written in high-level language and later replaced with the optimized ones

Operating System Structure

- **Monolithic** structure vs. componentized, modularized approach
  - component(module): well-defined portion of system with inputs, outputs and functions
- Simple Structure
  - Started simple and grew to become out-of-control
    - MS-DOS: applications can directly access I/O
    - Original UNIX (somewhat layered)



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

Kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

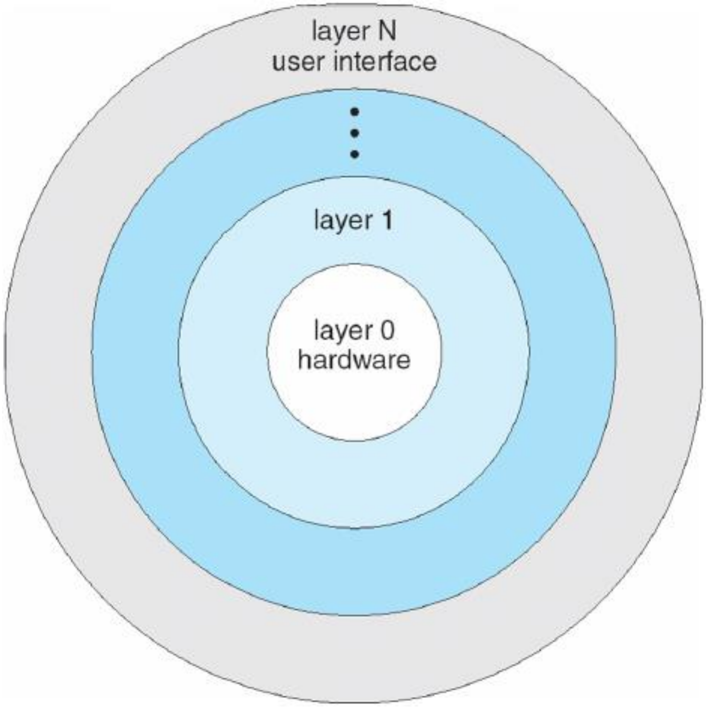
device controllers
disks and tapes

memory controllers
physical memory

Traditional UNIX system structure

OS Structure: Layered Approach

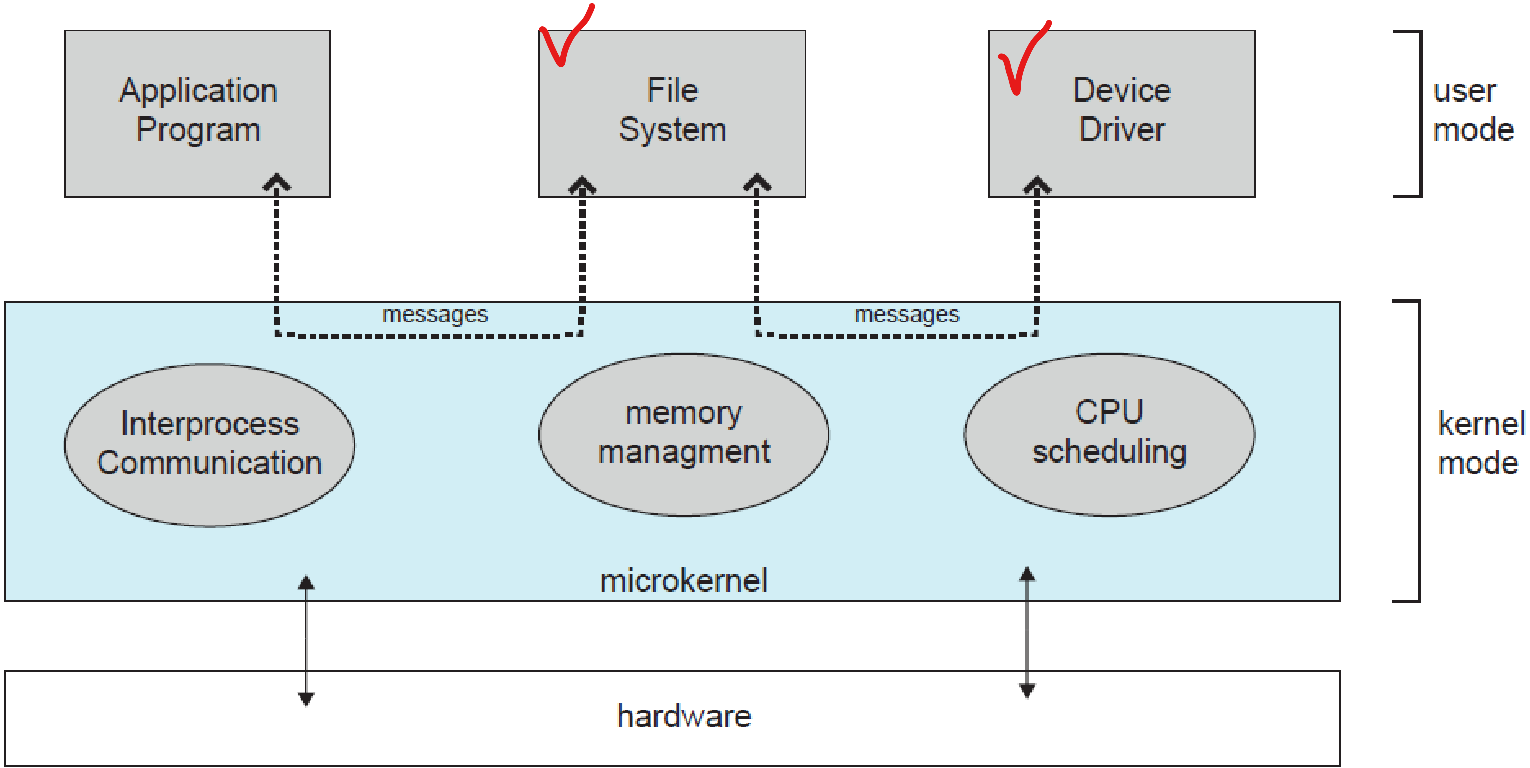
- Layered approach
  - OS is broken into a number of layers
    - bottom layer: layer 0 = hardware
    - Layer M = user interface
  - Layer M invokes operations of layer M-1
  - Layer M provides operations that can be called by layer M+1



- Advantage
  - simplicity → easy to debug
- Disadvantage
  - Defining layers is difficult
  - Less efficient
    - crossing layers may add overhead

Microkernels

- Key idea
  - Make the kernel minimal
  - Implement non-essential functions as user-level library
- Main function
  - Providing efficient communication between applications and services
- Advantage
  - highly extensible OS kernel
  - easy to port kernels
  - more secure and reliable
- Disadvantage
  - Impact on performance



Modules

- Loadable kernel modules
  - A way to extend kernel functionality dynamically
  - A module that can be loaded dynamically into kernel
  - Used to add support for new H/W (device drivers), new file systems or new system calls
- Without LKM, kernel has to be rebuilt every time new feature is added
- Kernel has to include everything
  - Larger footprint
  - Larger memory consumption
- In Linux,
  - modules have *.ko extension
  - modprobe: add/remove modules
  - lsmod: show the status of modules
  - modinfo: show module information

Modules

- lsmod

```
Module                  Size   Used by
rfcomm                  69632  2
xt_CHECKSUM             16384  1
iptable_mangle          16384  1
ipt_MASQUERADE          16384  4
nf_nat_masquerade_ipv4  16384  1 ipt_MASQUERADE
ipt_REJECT              16384  2
nf_reject_ipv4          16384  1 ipt_REJECT
xt_tcpudp               16384  6
xfrm_user               32768  1
xfrm_algo               16384  1 xfrm_user
iptable_nat             16384  1
nf_conntrack_ipv4       16384  3
nf_defrag_ipv4          16384  1 nf_conntrack_ipv4
nf_nat_ipv4             16384  1 iptable_nat
xt_addrtype             16384  2
ebtable_filter          16384  0
ebtables                36864  1 ebtable_filter
xt_conntrack            16384  2
nf_nat                  24576  2 nf_nat_ipv4,nf_nat_masquerade_ipv4
br_netfilter            24576  0
```

- modinfo

```
~$ modinfo nf_nat_ipv4
filename:       /lib/modules/4.4.0-66-generic/kernel/net/ipv4/netfilter/nf_nat_ipv4.ko
alias:          nf-nat-2
license:        GPL
srcversion:     62E44ADA1A1F6F29B5E6B7A
depends:        nf_nat,nf_conntrack
intree:         Y
vermagic:       4.4.0-66-generic SMP mod_unload modversions
```

OS Debugging

- Problem types
  - Fail-stop problem
  - Performance problem
- Failure analysis
  - Applications: log files
  - OS kernel: core (memory) dump
- Performance Tuning
  - Identify and remove the bottleneck