Operating Systems
CS2006

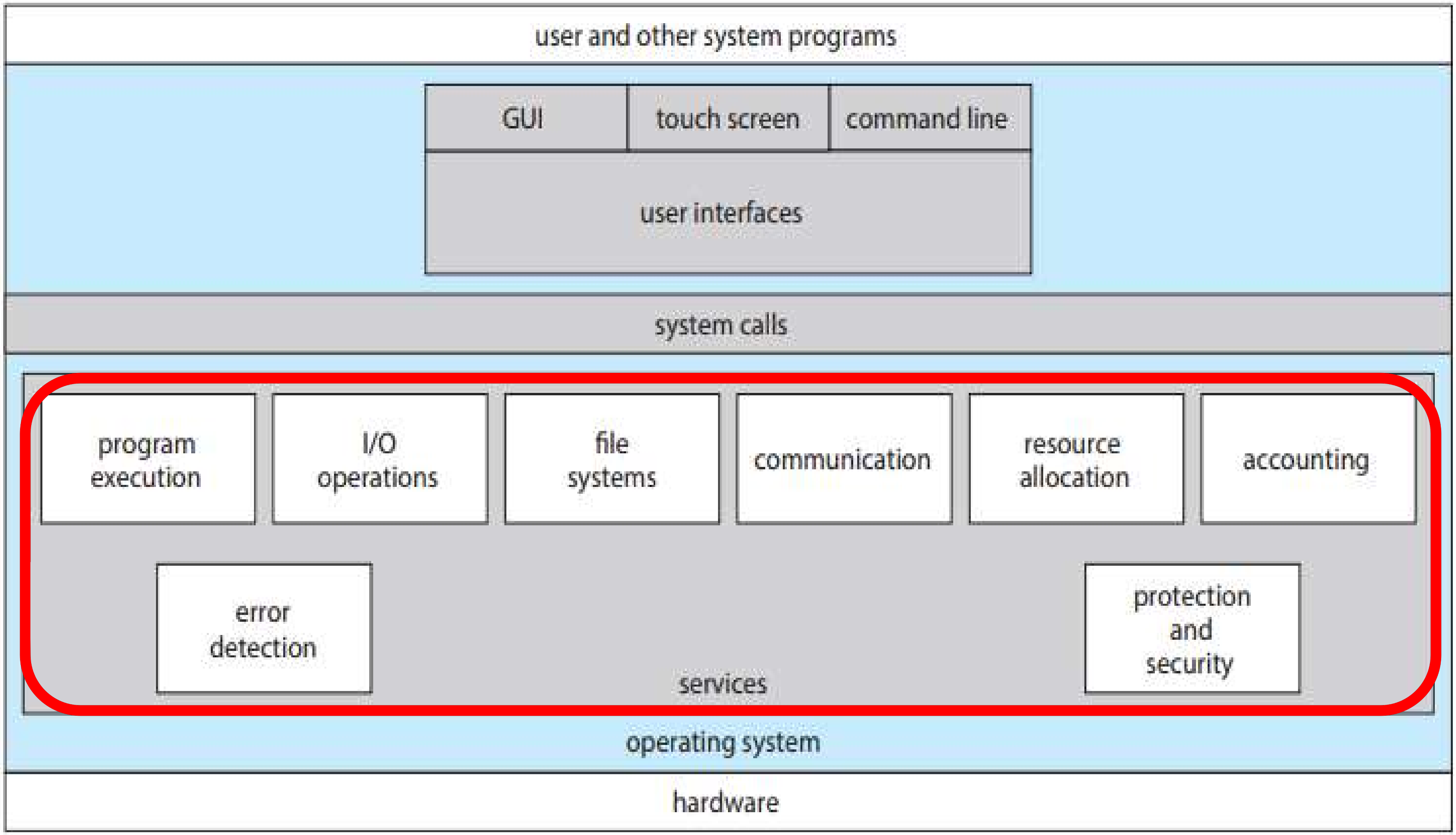
Lecture 3

System Calls & OS Structure

6th February 2023

Dr. Rana Asif Rehman

A View of Operating System Services



What Categories of Services Might be Provided by OS?

- Program execution
- OS Operations
- Process management
- Memory management
- Storage management
- I/O Subsystem
- Protection & Security

1. Program Execution

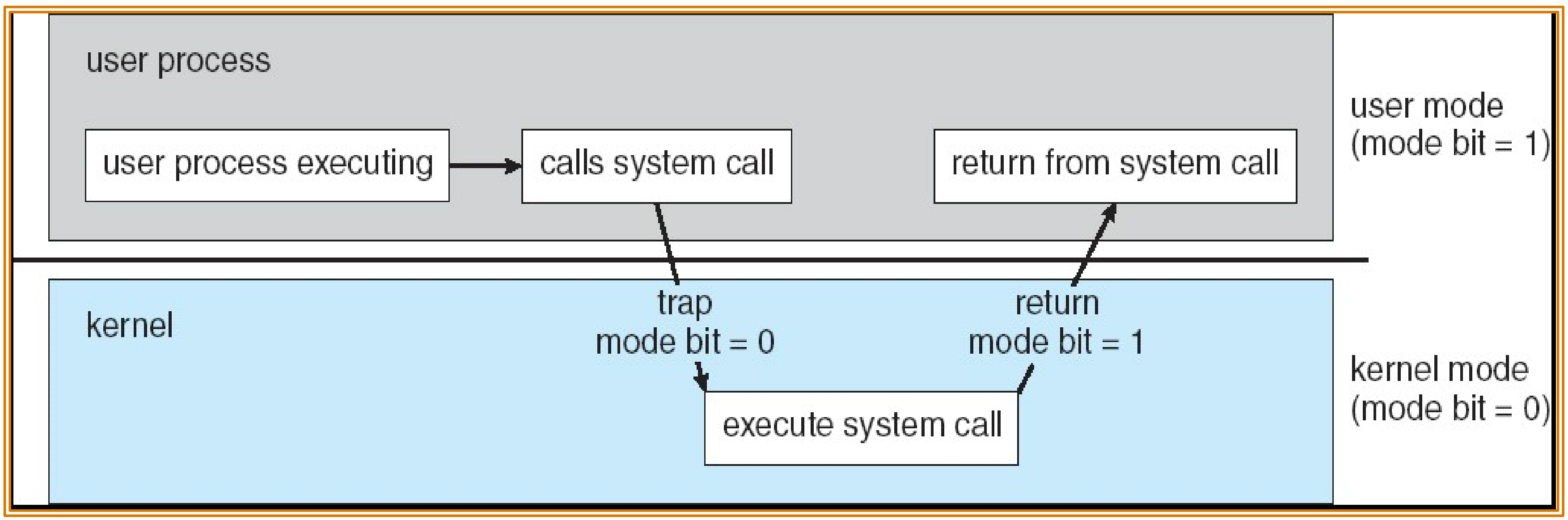
- **Multiprogramming** needed for efficiency of utilization
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which the CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be << 1 second
  - Each user has at least one program executing in memory ➔ **process**
  - If several jobs ready to run at the same time ➔**CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

2. OS Operations

- Interrupt driven by hardware
- Software error or request creates exception or trap
  - Division by zero, request for operating system service
  - Other process problems include infinite loop, processes modifying each other or the operating system
  - Dual-mode operation allows OS to protect itself and other system components
- User mode and kernel mode
  - Mode bit provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as privileged, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter reaches zero, generate an interrupt
  - Setup before scheduling process to regain control or terminate program that exceeds allotted time



3. Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaiming of any reusable resources
- Single-threaded process has one program counter specifying the location of the next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system, running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process communication
- Providing mechanisms for process synchronization
- Providing mechanisms for deadlock handling

4. Memory Management

- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
- All data in memory before and after processing
- All instructions in memory to allow execution
- Memory management determines what is in memory
  - Optimizing CPU utilization & computer response to users

5. Storage Management

5.1. File-System management

- Files usually organized in directories
- Access control on most systems to determine who can access what
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit – file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
  - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

5. Storage Management

5.2. Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Disk fragmentation / defragmentation
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed – by OS or applications
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

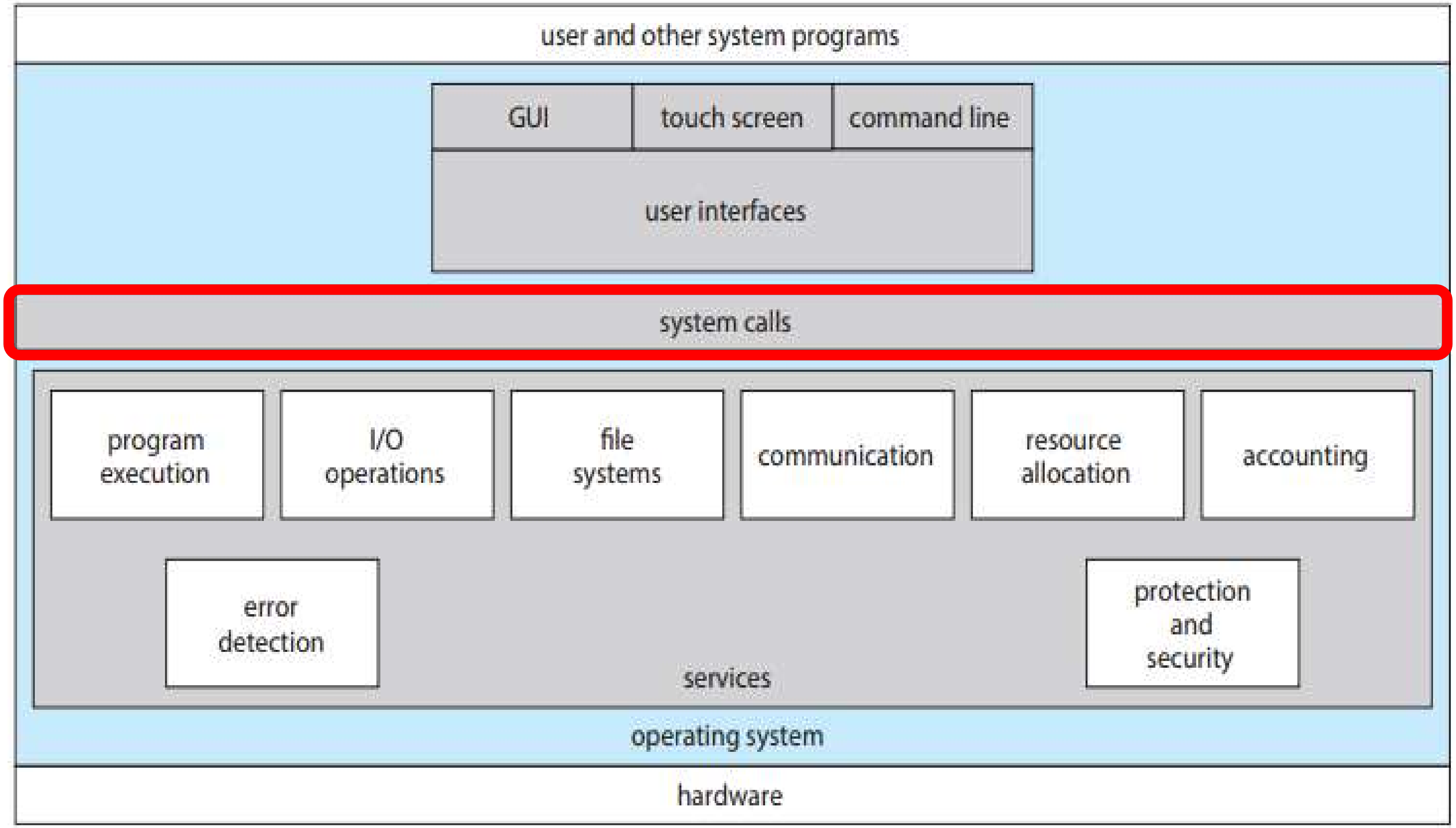
6. I/O Subsystem

- I/O Subsystem responsible for
  - Memory management of I/O including
    - Buffering: storing data temporarily while it is being transferred
    - Caching: storing parts of data in faster storage for performance
    - Spooling: the overlapping of output of one job with input of other jobs
  - General device-driver interface
  - Drivers for specific hardware devices
- One purpose of OS is to hide peculiarities of hardware devices from the user

7. Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS
- Security – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to an effective ID with more rights

System Calls

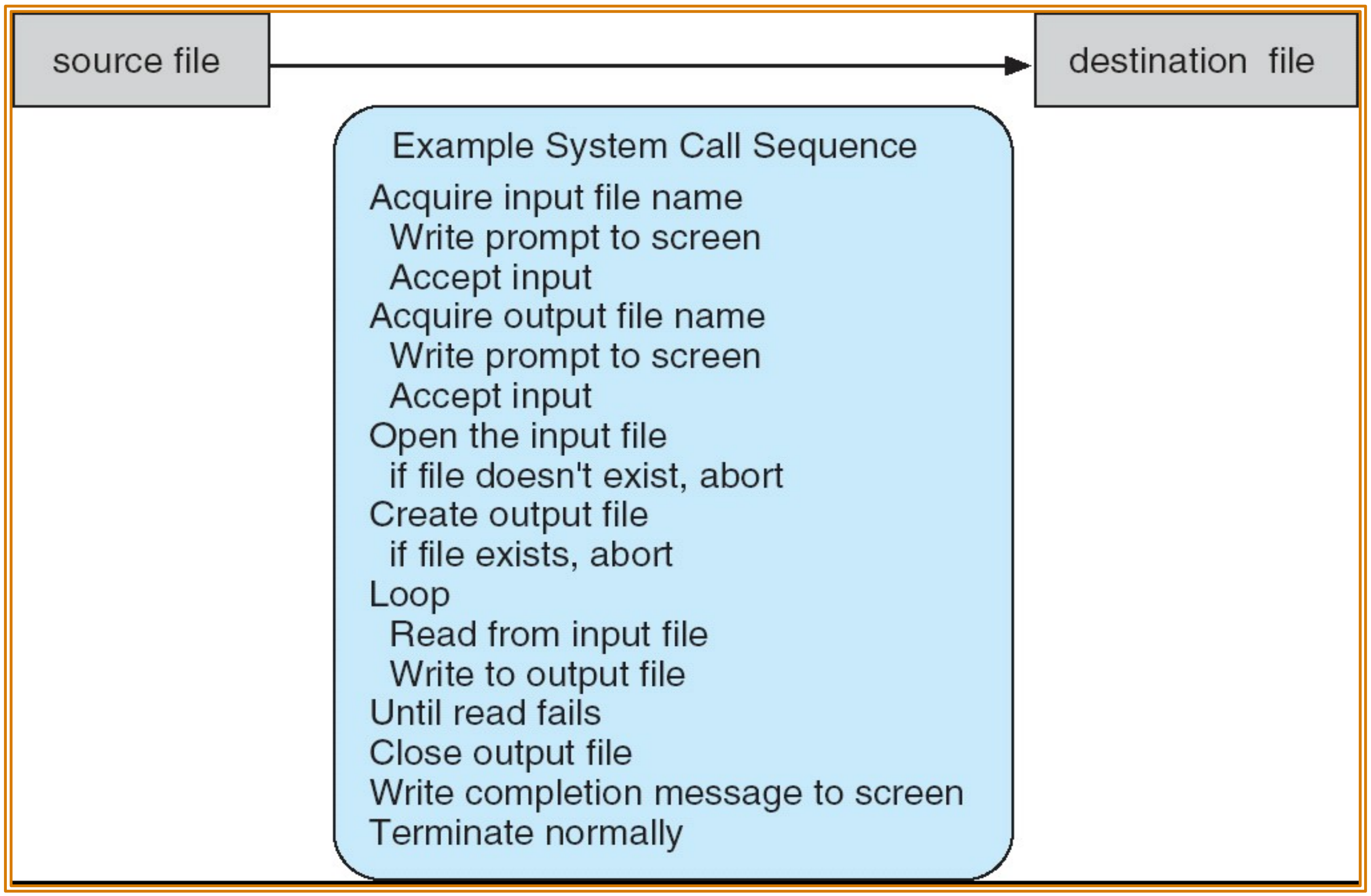


System Calls

- Programming interface to the services provided by OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

return value

ReadFile(): Win32 API—a function for reading from a file

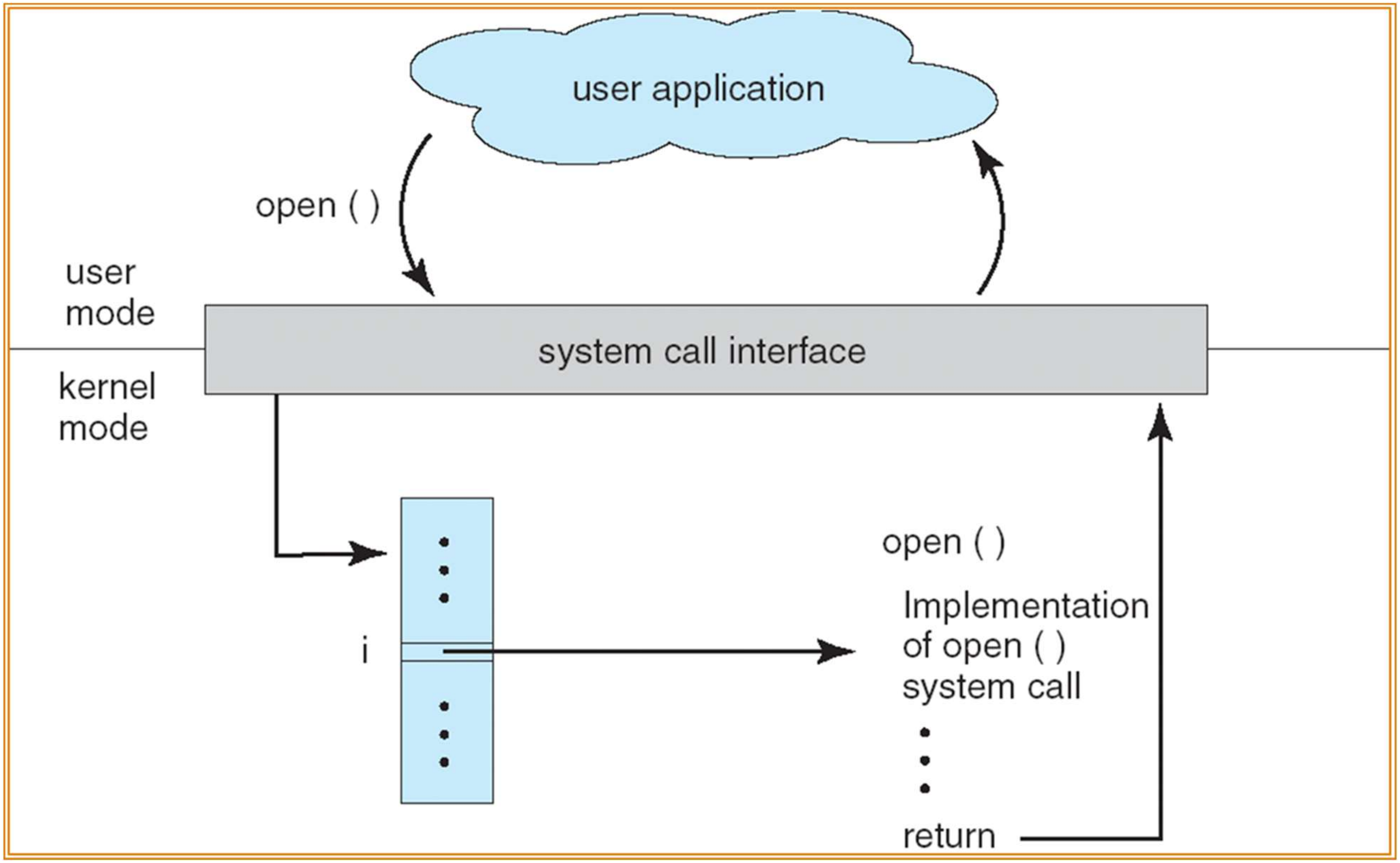
```
BOOL   ReadFile c  (HANDLE          file,
                    LPVOID          buffer,
                    DWORD           bytes To Read,
                    LPDWORD         bytes Read,
                    LPOVERLAPPED    ovl);
```

parameters

function name

- A description of the parameters passed to ReadFile()
  - HANDLE file—the internal handle of the file to be read
  - LPVOID buffer---A buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

API – System Call – OS Relationship

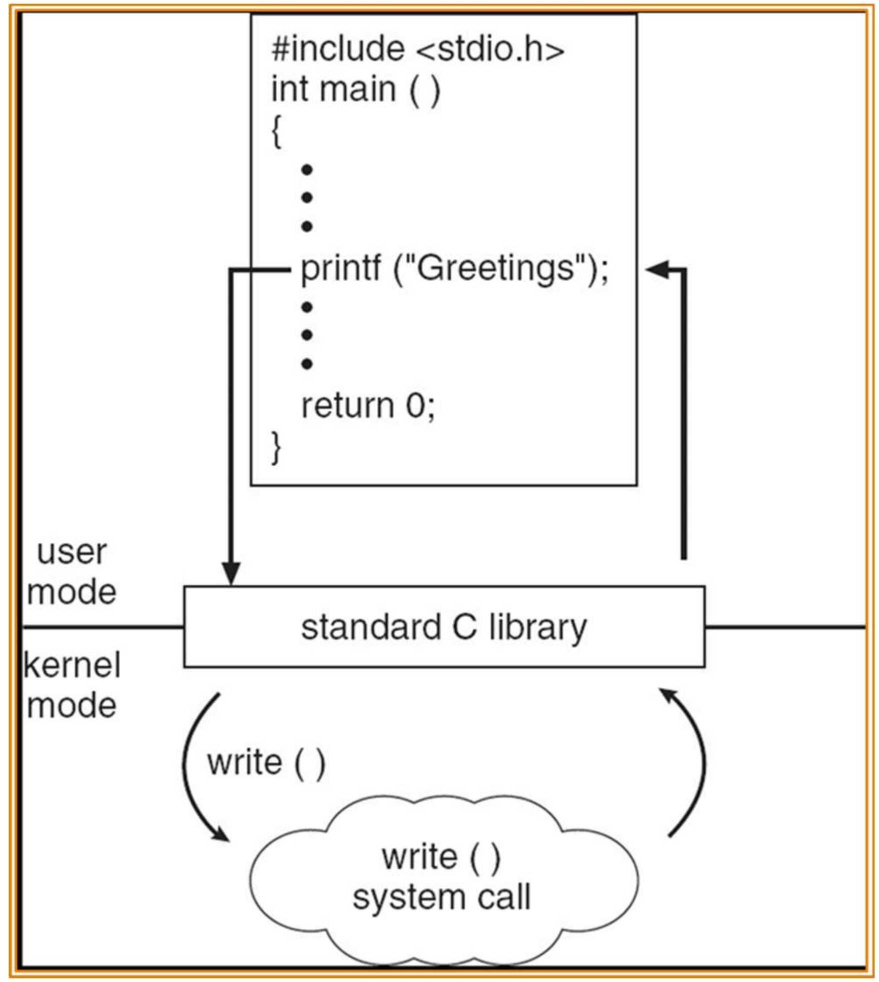


System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result of the call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

Standard C Library Example

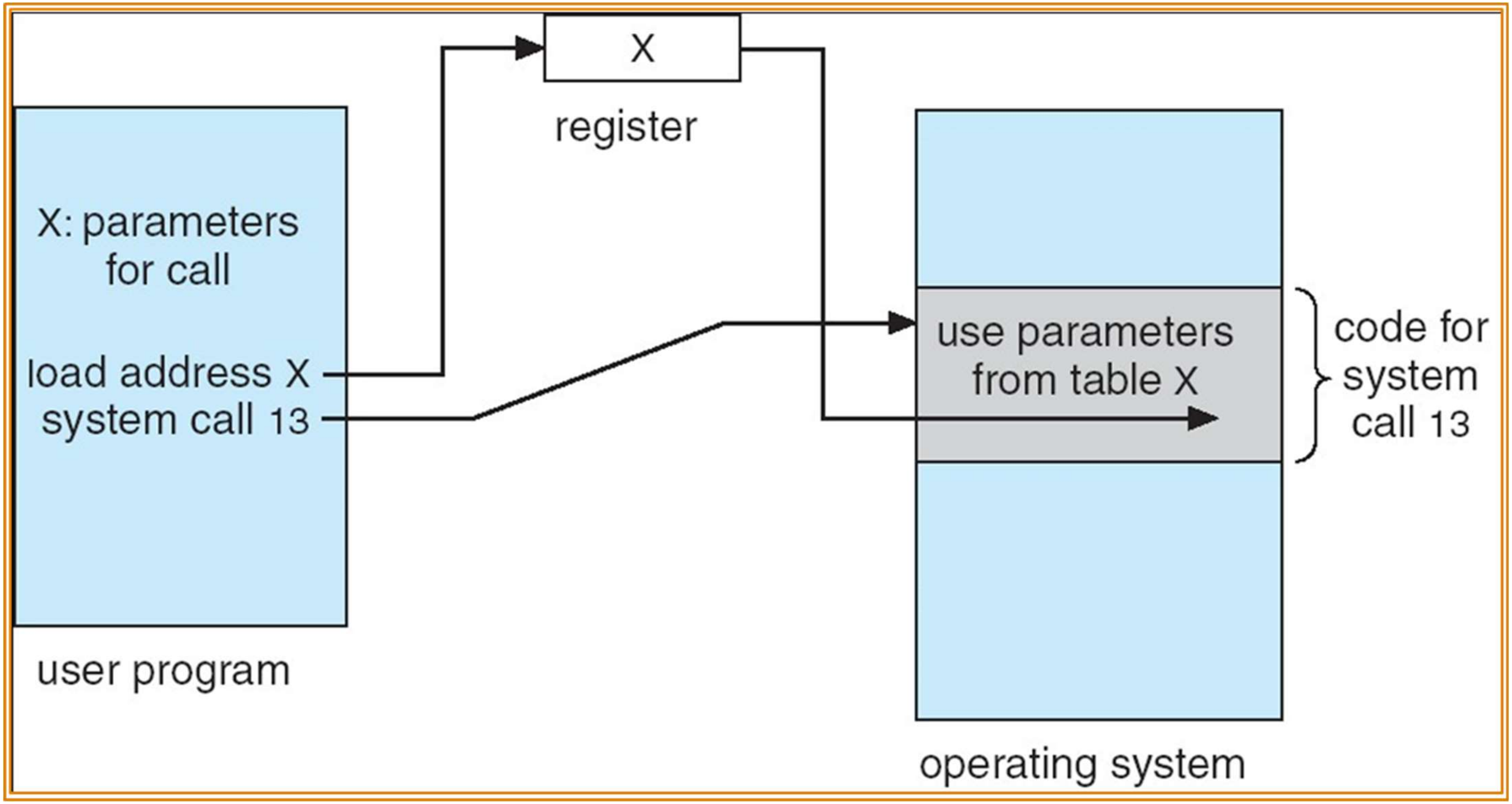
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Methods used to pass parameters to OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed*, onto the *stack* by the program and popped off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

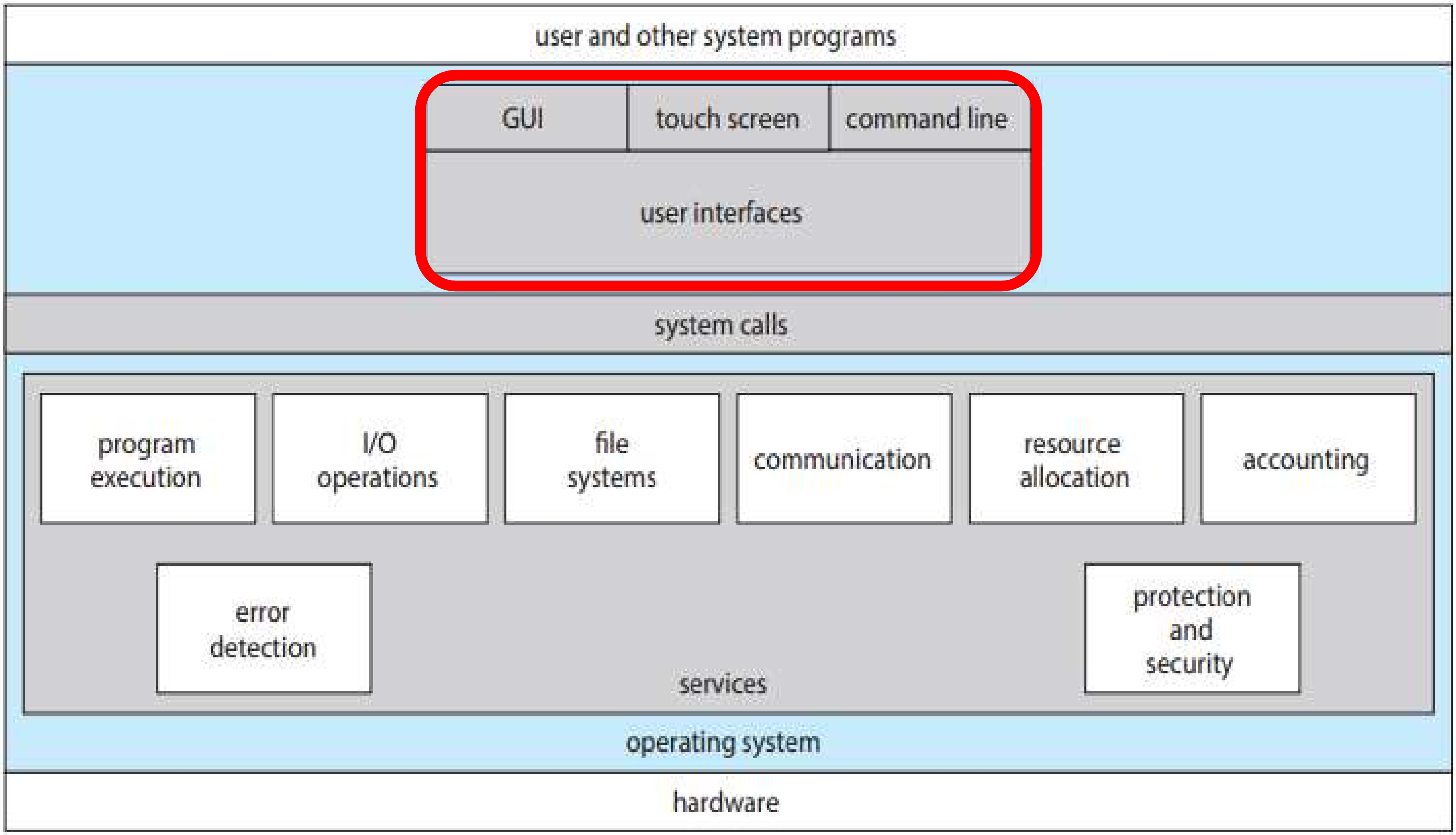
Parameter Passing via Table



Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

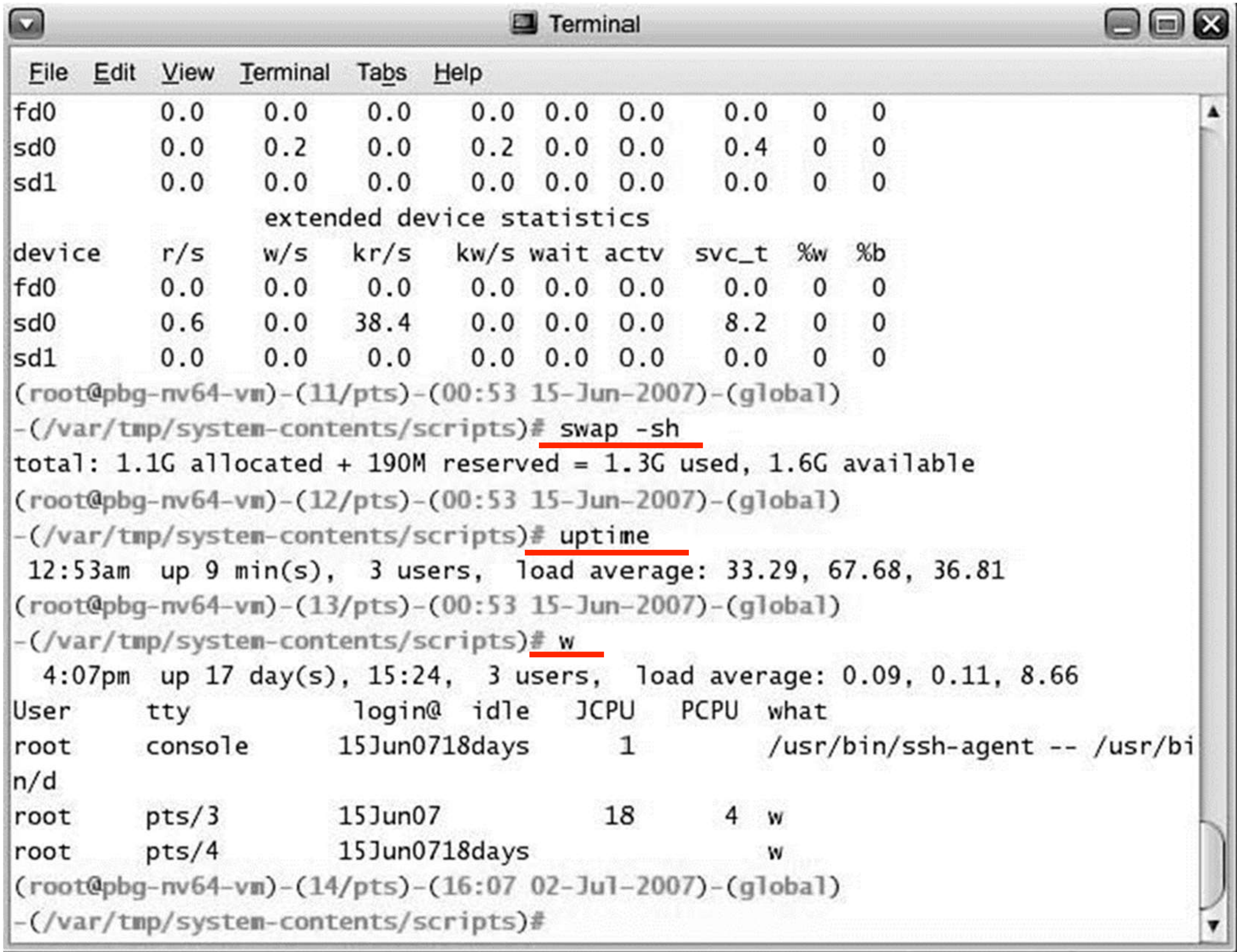
User Interface



User Operating System Interface - CLI

- Command-Line Interface (CLI) allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple flavors implemented – shells
- Primarily receives command from user and executes it
  - Sometimes commands built-in, sometimes just names of programs, sometimes a combination
  - If the latter, adding new features doesn't require CLI modification

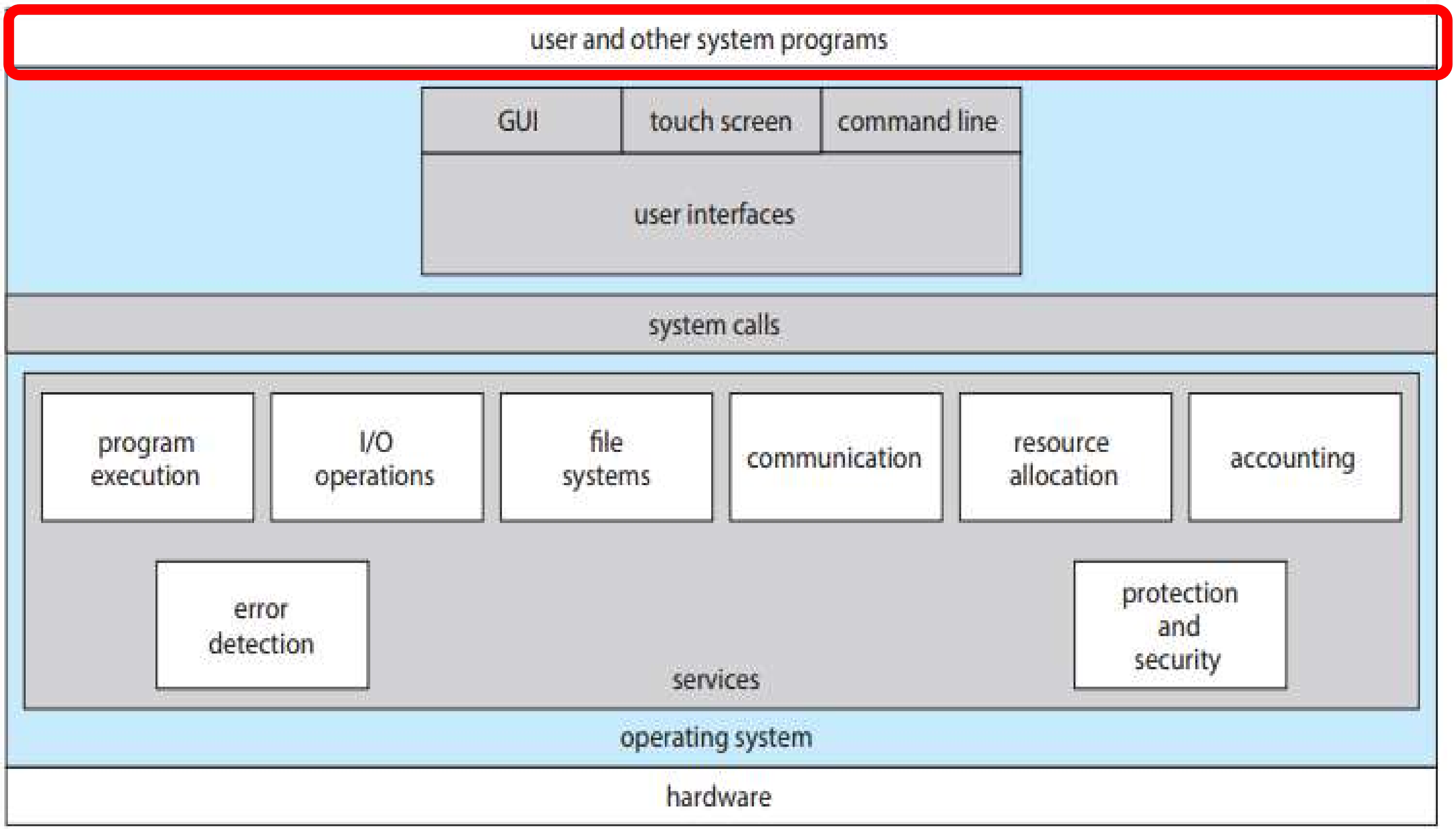
Bourne Shell Command Interpreter



User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI; CLI is "command" shell
  - Apple Mac OS X has "Aqua" GUI; UNIX kernel underneath and multiple shells available
  - Solaris has multiple GUIs; CLI is multiple shells

User and System Programs



System Programs

- System programs provide a convenient environment for program development and execution. Including:
  - File manipulation
  - Status information
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls

OS Structure

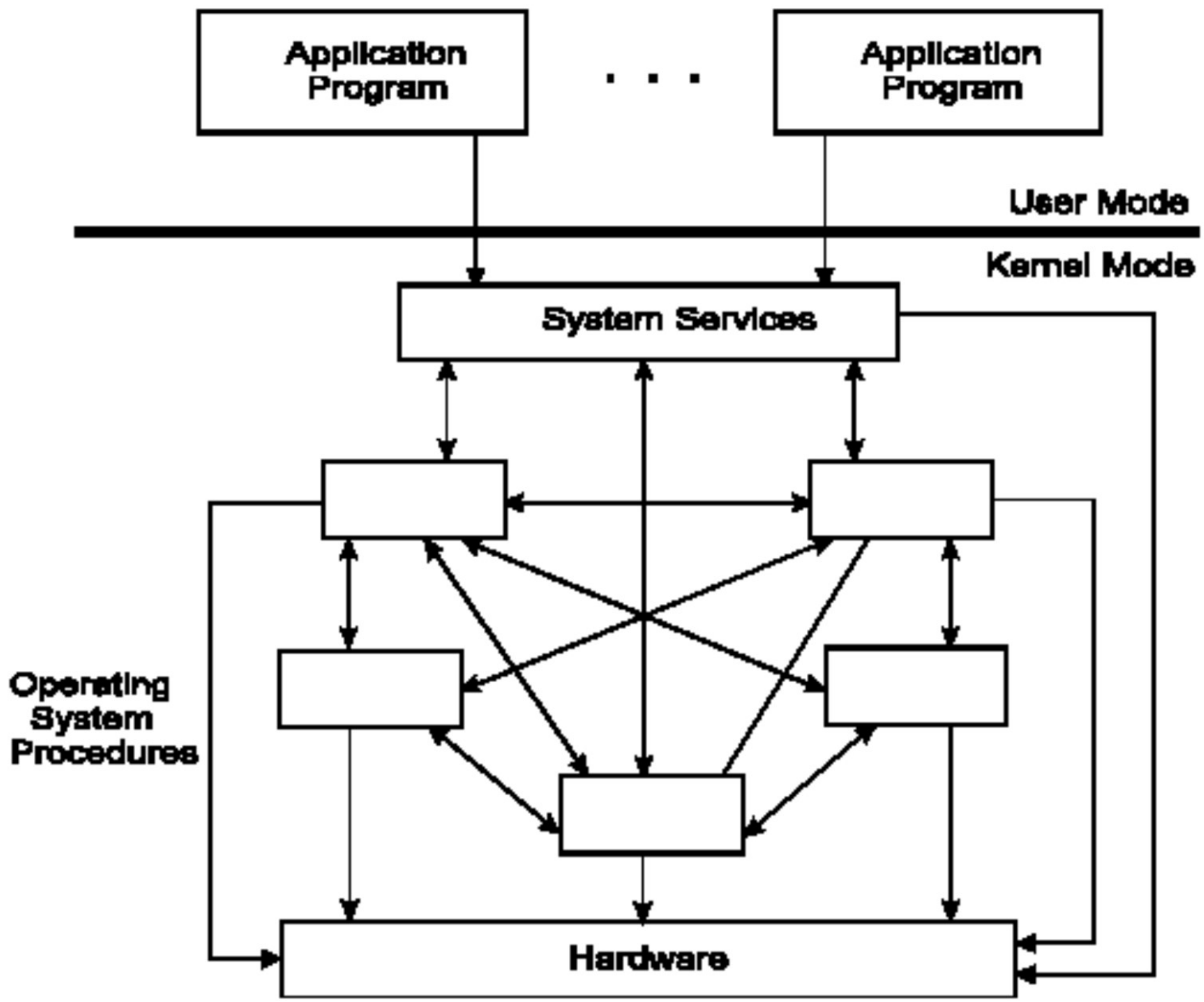
Operating System Design and Implementation

- Affected by choice of hardware, type of system
- *User* goals and *System* goals
  - ***User goals*** – operating system should be convenient to use, easy to learn, reliable, safe, secure, and fast
  - ***System goals*** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error free, secure, and efficient
- **Important principle is separation**
  - ***Policy***: What will be done?
  - ***Mechanism***: How to do it?
  - The separation of policy from mechanism is a very important principle, it allows maximum ***flexibility*** if policy decisions are to be changed later

Operating Systems Structures

- Structure/Organization/Layout of OSs:
  - Monolithic (one unstructured program)
  - Layered
  - Microkernel
  - Kernel Modules
  - Virtual Machines
- The role of Virtualization

1. Monolithic Operating System



Monolithic OS – Basic Structure

- Application programs that invoke the requested system services.
- A set of system services that carry out the operating system procedures/calls.
- A set of utility procedures that help the system services.

MS-DOS System Structure

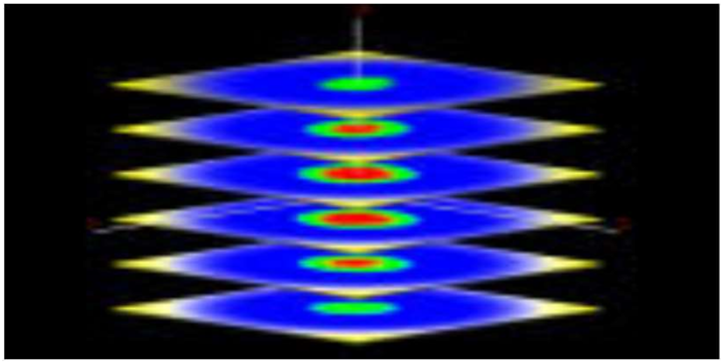
- MS-DOS – written to provide functionality in the least space:
  - not divided into modules (monolithic).
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

UNIX System Structure

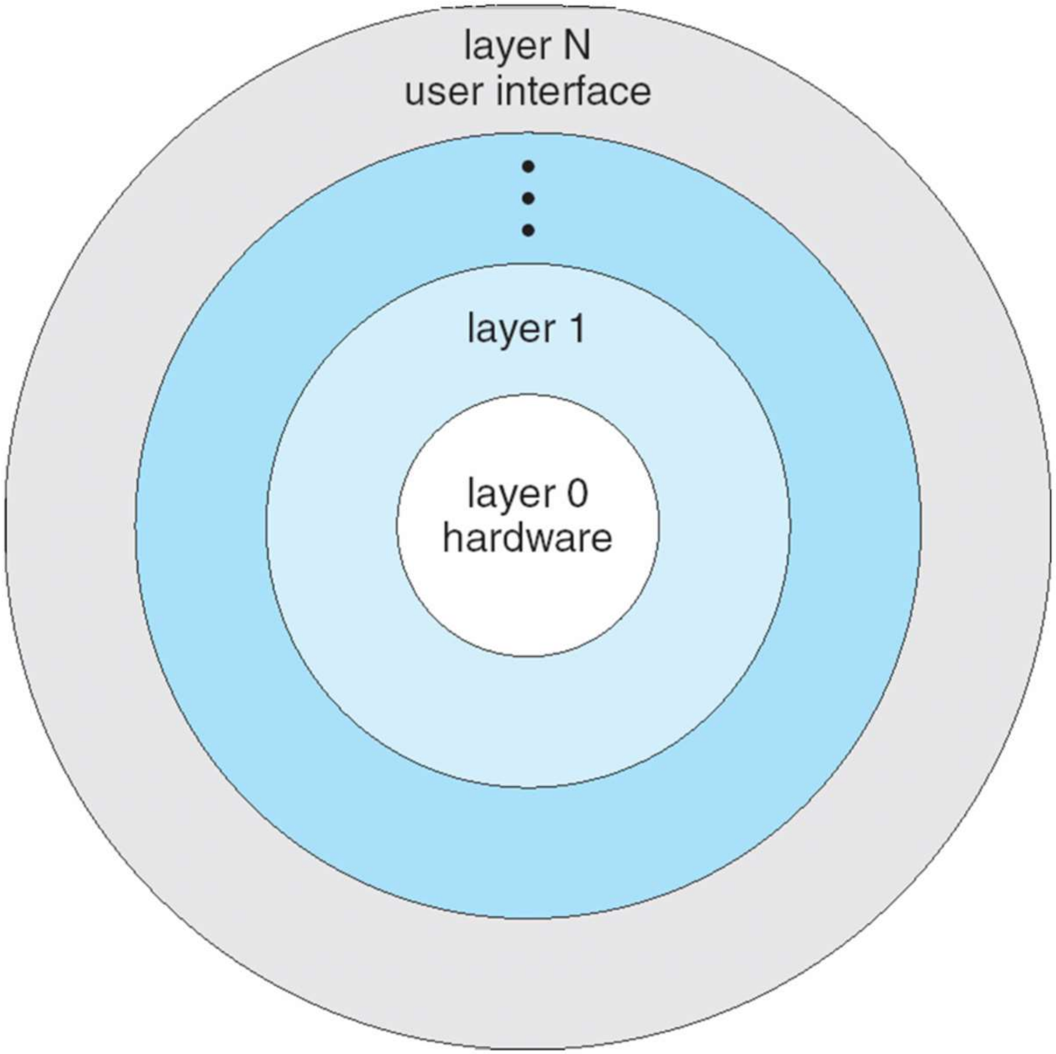
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
- **The kernel**
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level
- **Systems programs**

2. Layered Approach

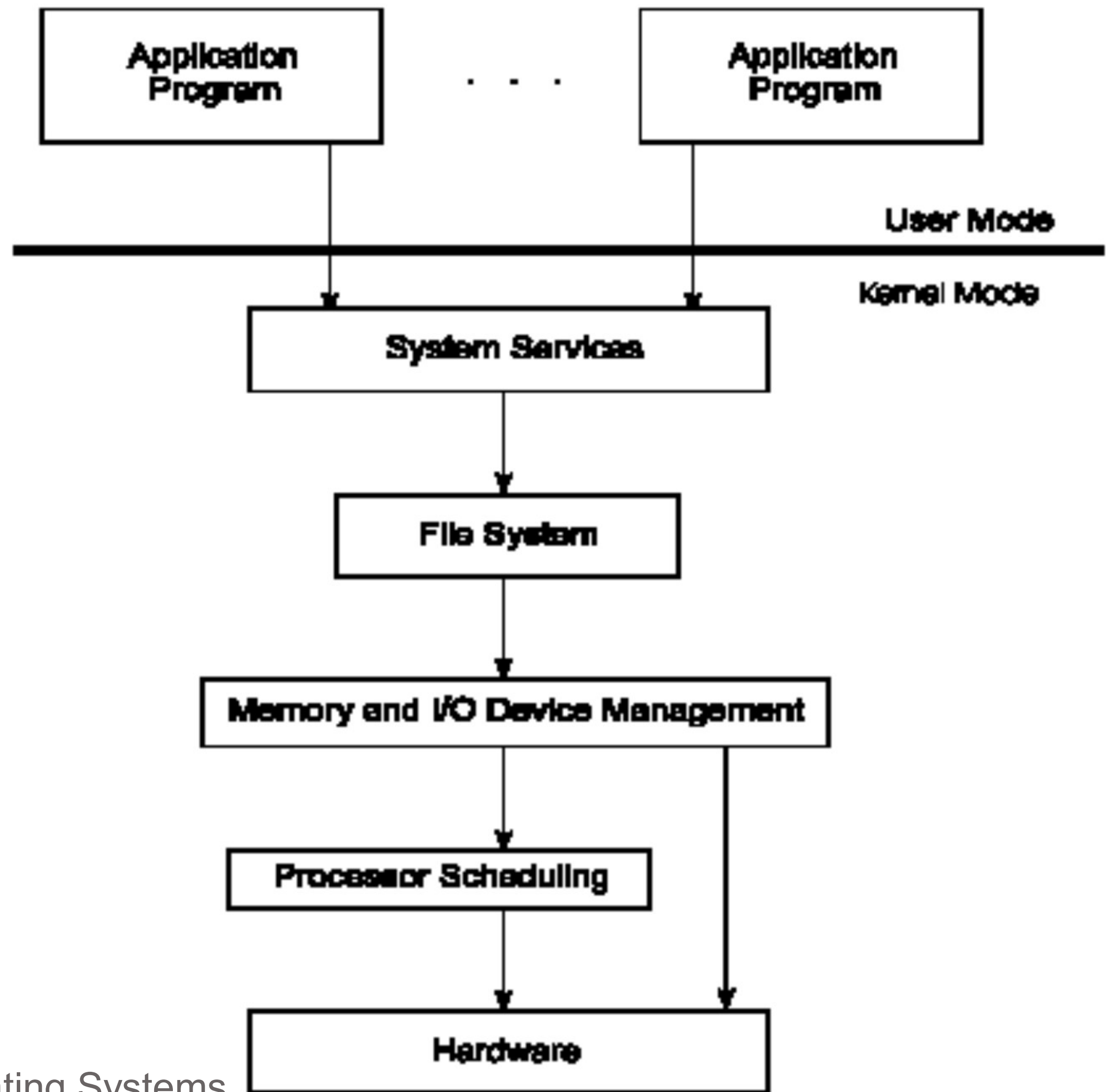
- The operating system is divided into a number of layers (levels), each built on top of lower layers
- The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



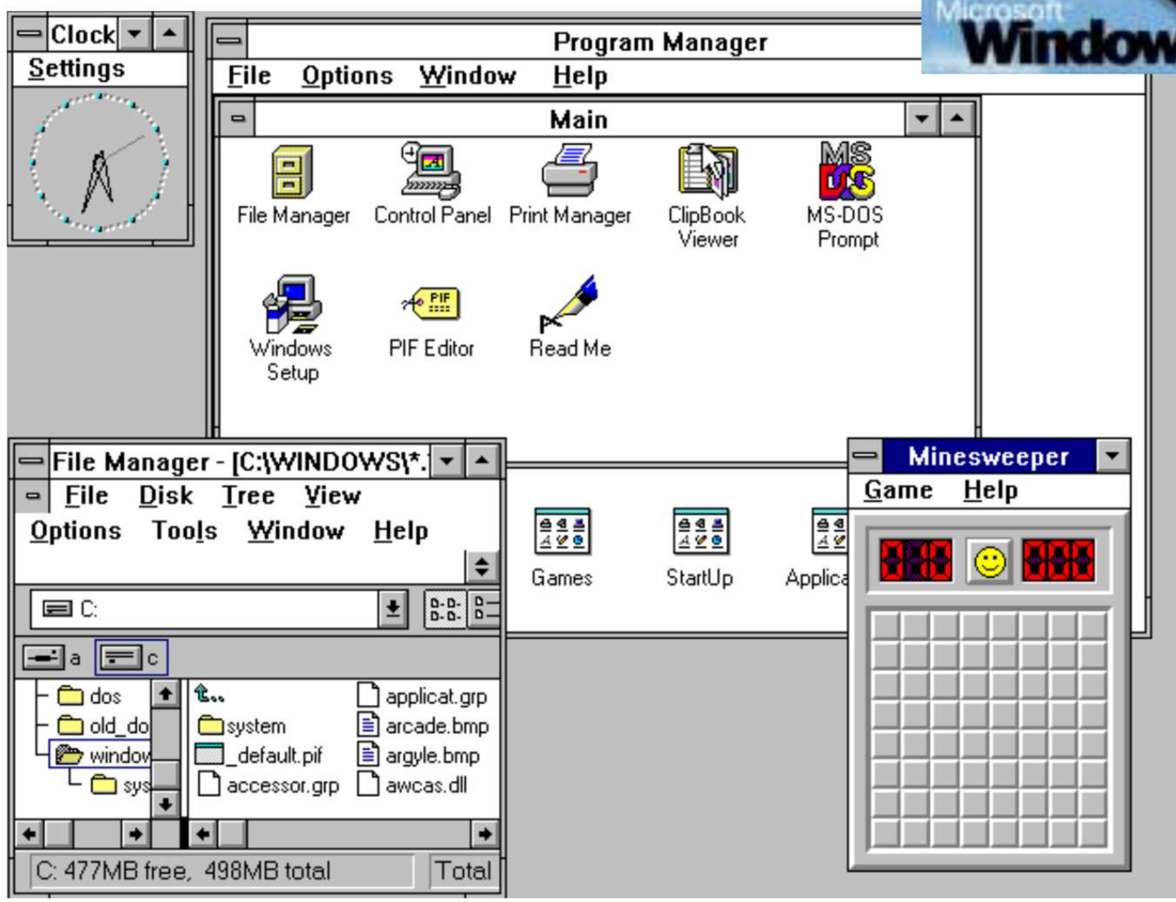
Layered Operating System



Operating System Layers



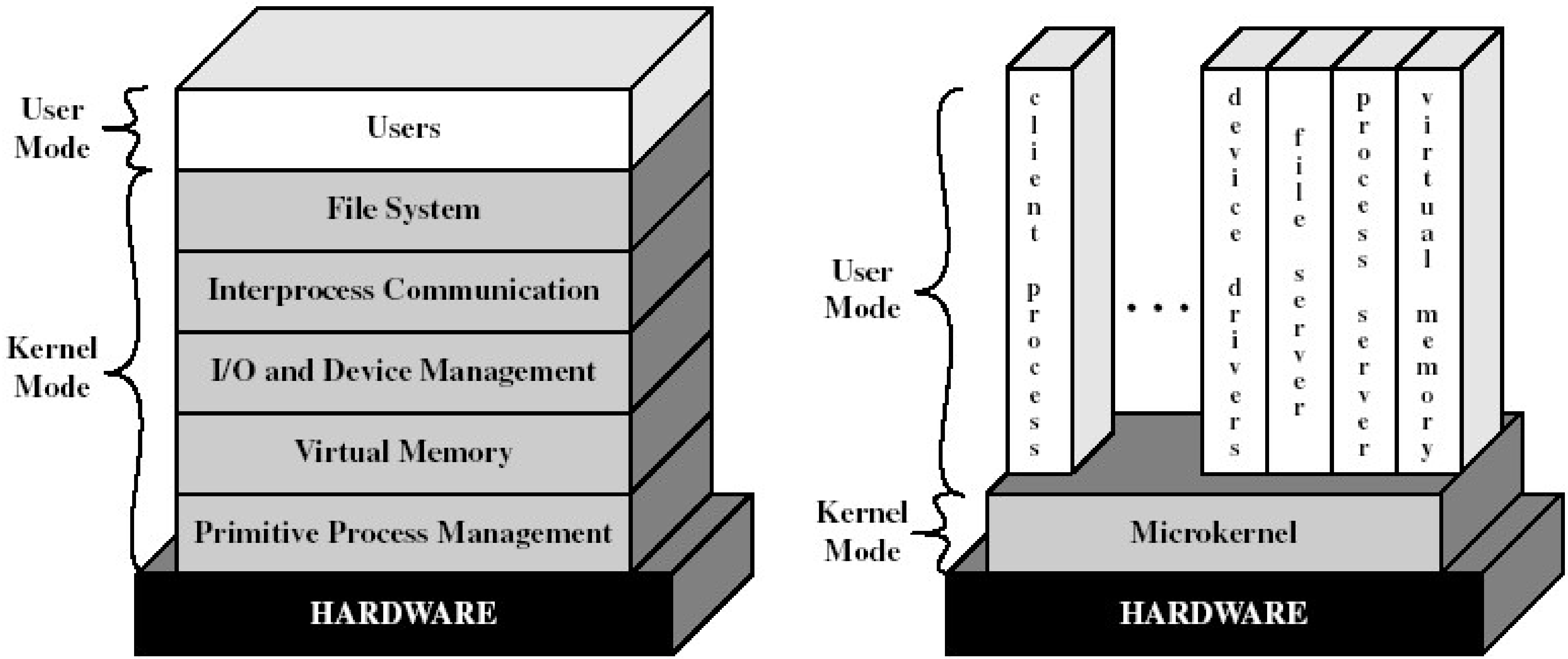
Older Windows System Layers



3. Microkernel System Structure

- Move as much functionality as possible from the kernel into "user" space.
- Only a few essential functions in the kernel:
  - primitive memory management (address space)
  - I/O and interrupt management
  - Inter-Process Communication (IPC)
  - basic scheduling
- Other OS services are provided by processes running in user mode (vertical servers):
  - device drivers, file system, virtual memory…

Layered vs. Microkernel Architecture



(a) Layered kernel

(b) Microkernel

Benefits of a Microkernel Organization

- Extensibility/Reliability
  - modular design
  - easier to extend a microkernel
  - more reliable (less code is running in kernel mode)
  - more secure (less code to be validated in kernel)
  - small microkernel can be rigorously tested
- Portability
  - changes needed to port the system to a new processor is done in the microkernel, not in the other services.

Mach 3 Microkernel Structure

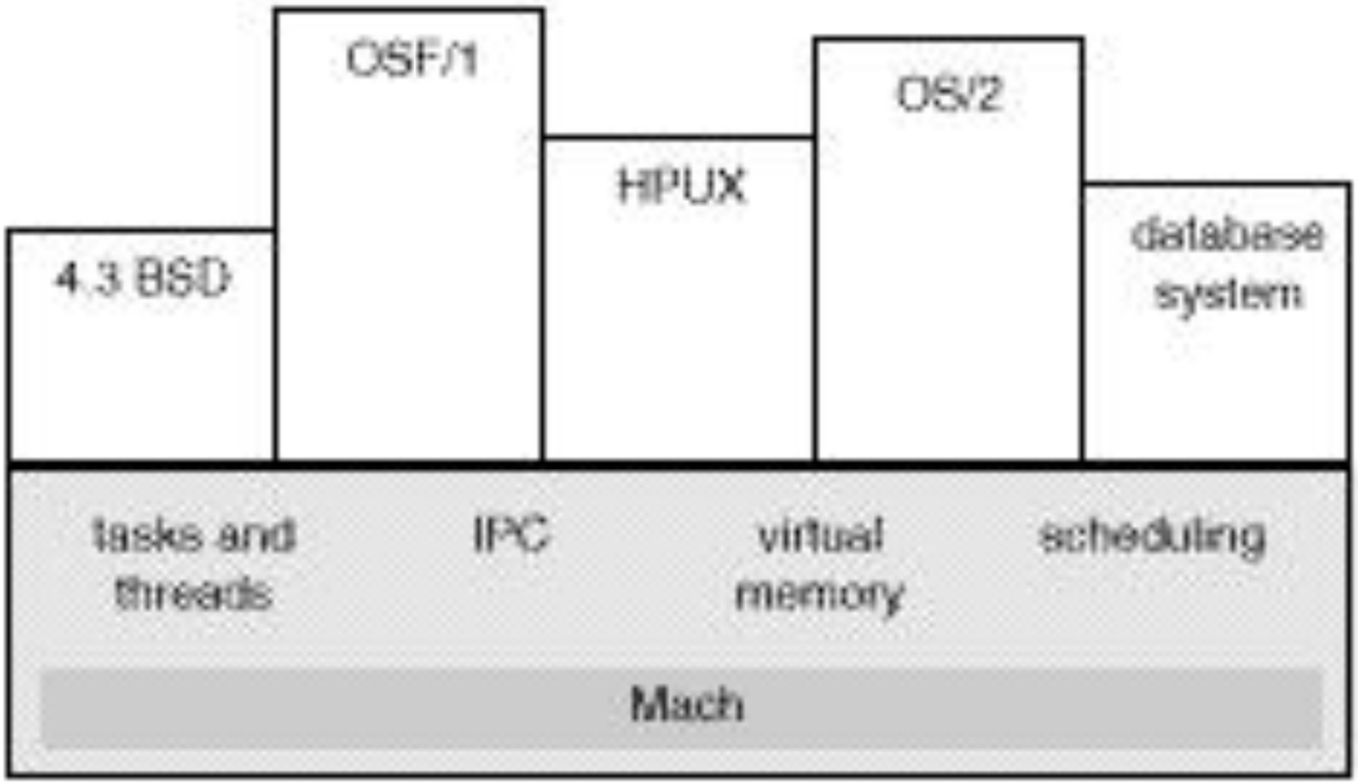
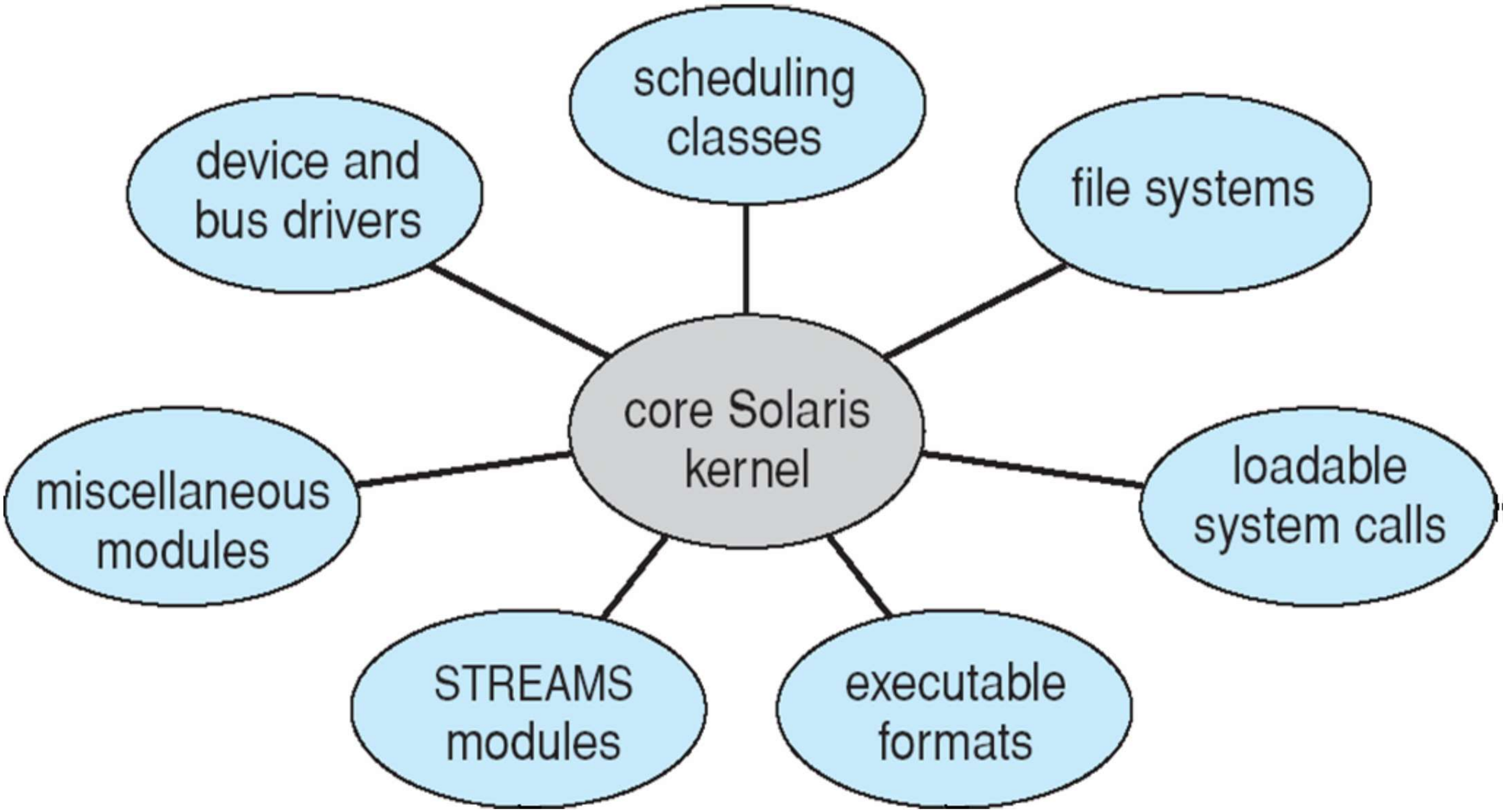


Figure A.1 Mach 3 structure.

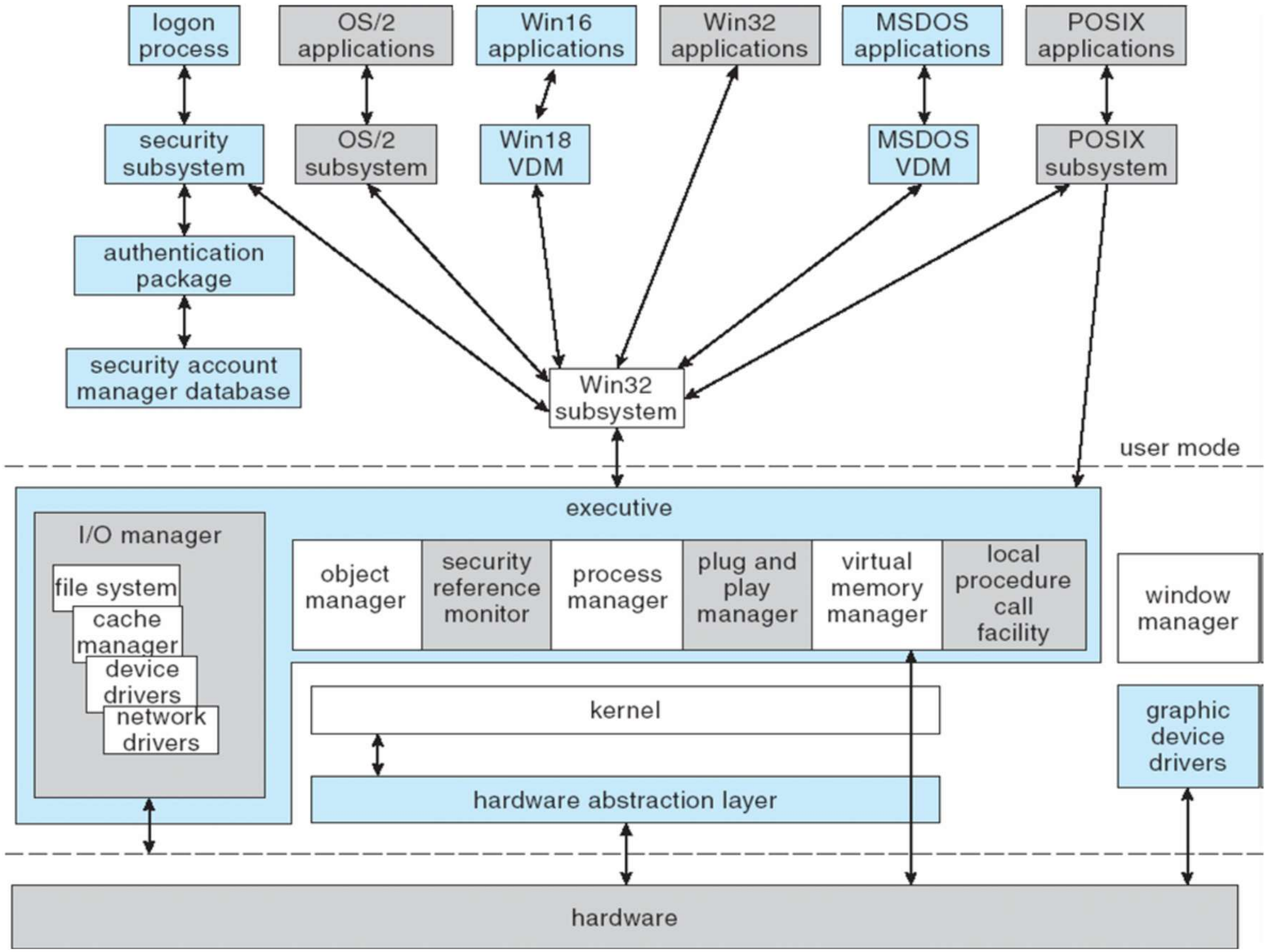
4. Kernel Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but more flexible

Solaris Modular Approach



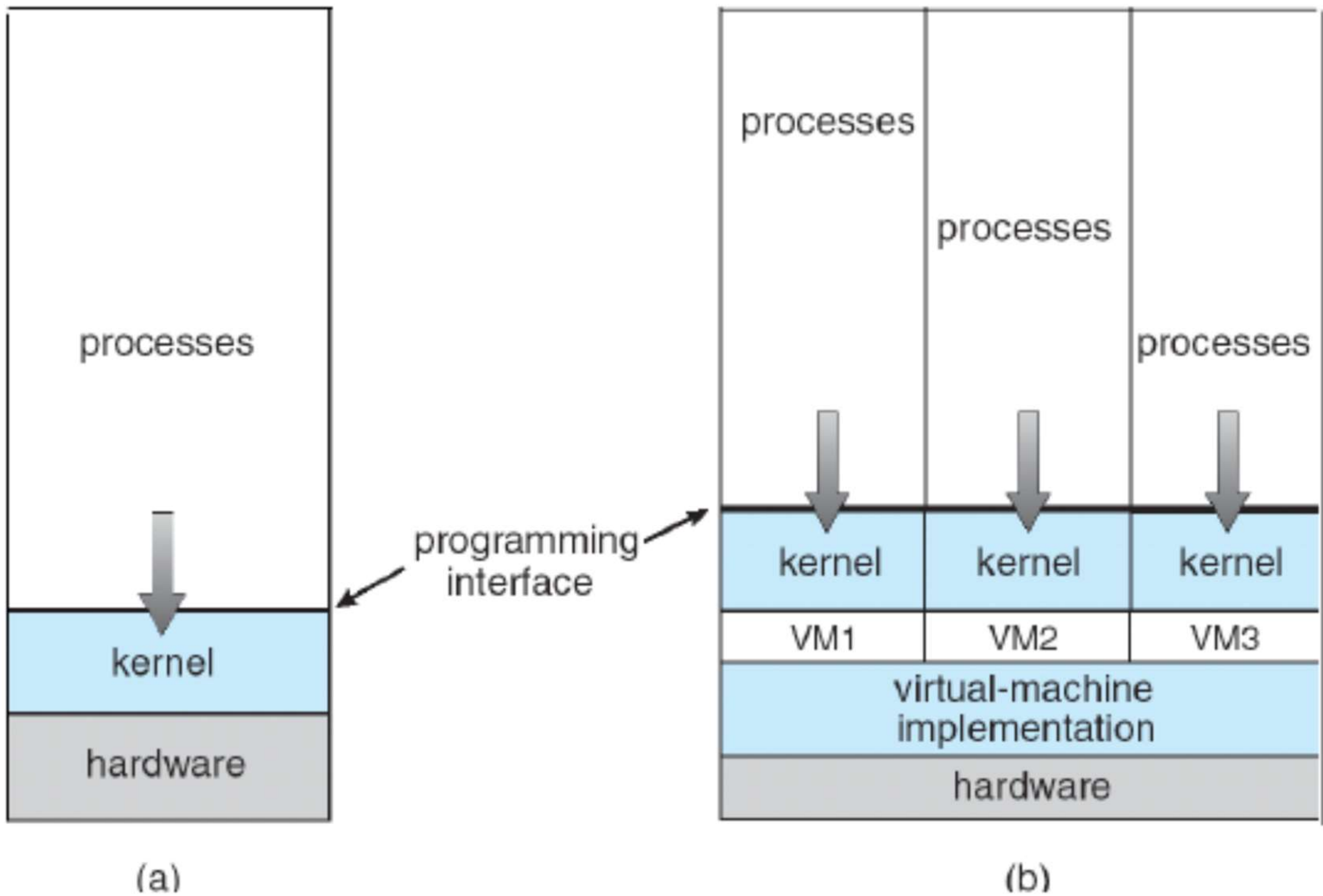
XP Architecture ?



5. Virtual Machines

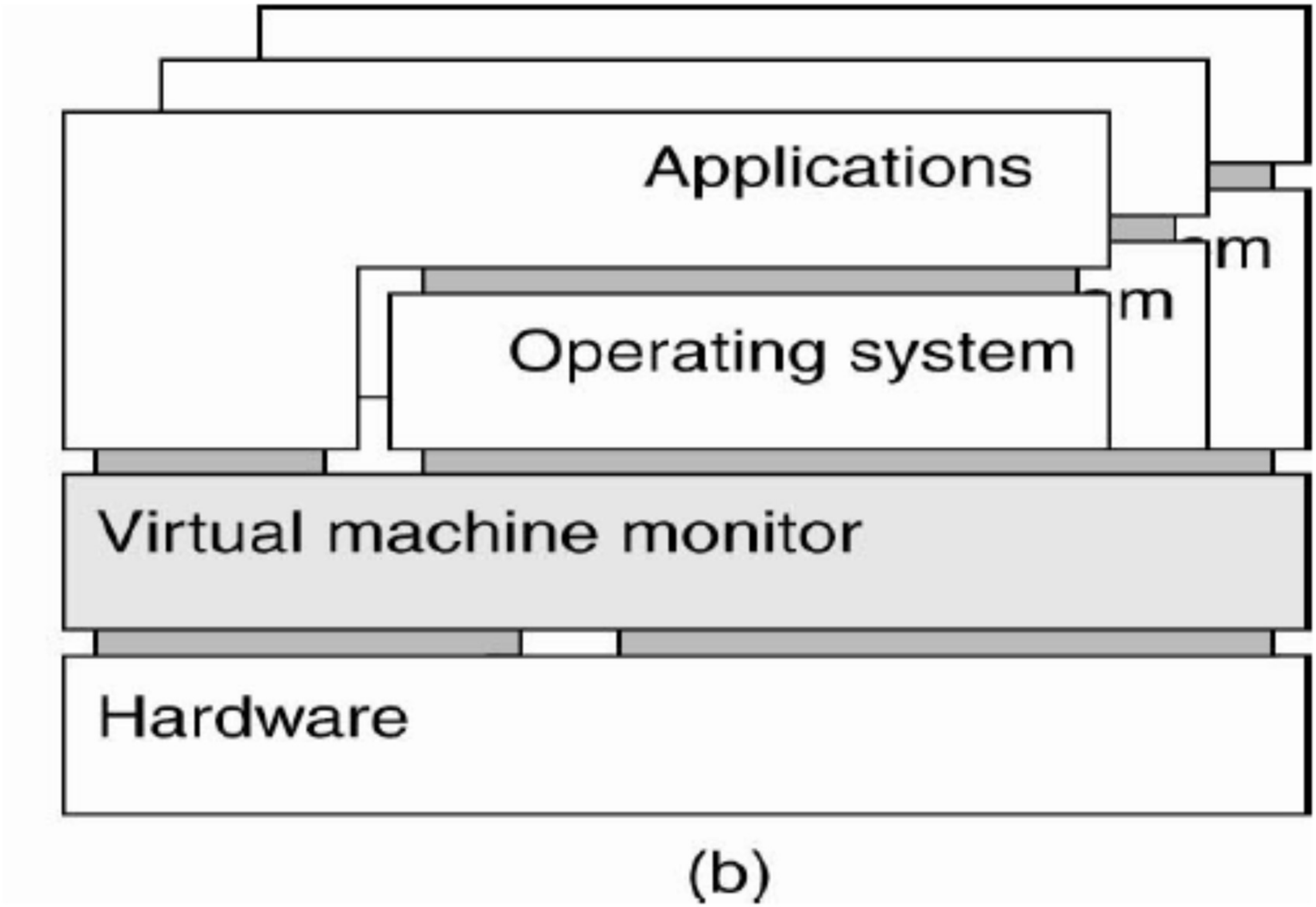
- A **virtual machine** takes the layered approach to its logical next step. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The operating system **host** creates the illusion that a process has its own processor (and virtual memory)
- Each **guest** provided with a (virtual) copy of underlying computer

Virtual Machines (Cont.)

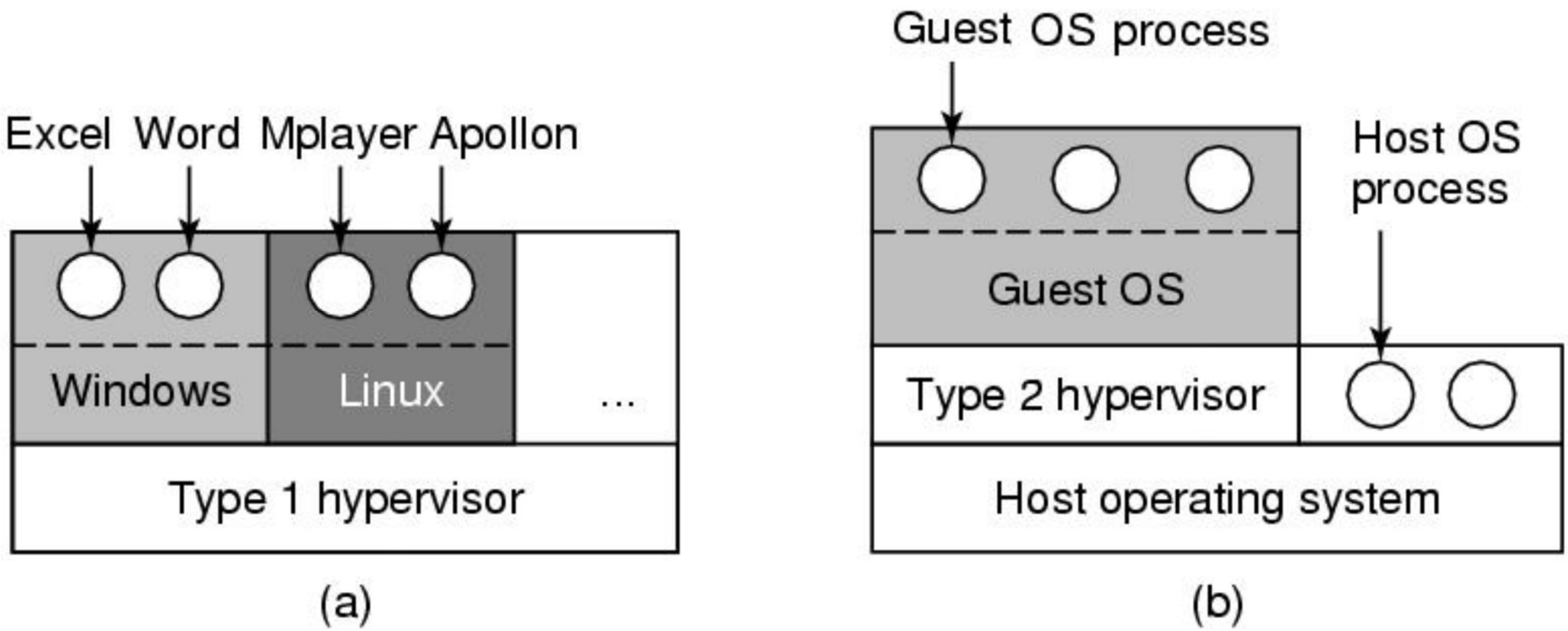


(a) Non-virtual machine (b) virtual machine

Hypervisor / VMM



Types of Hypervisors



(a) A type 1 hypervisor

(b) A type 2 hypervisor

Para- vs. Full-virtualization

- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardware
- Guest can be an OS, or in the case of Solaris 10 applications running in containers
- Full-virtualization: unmodified guest OSes

References

- Operating System Concepts (Silberschatz, 9th edition) Chapter 1, 2.1-2.5