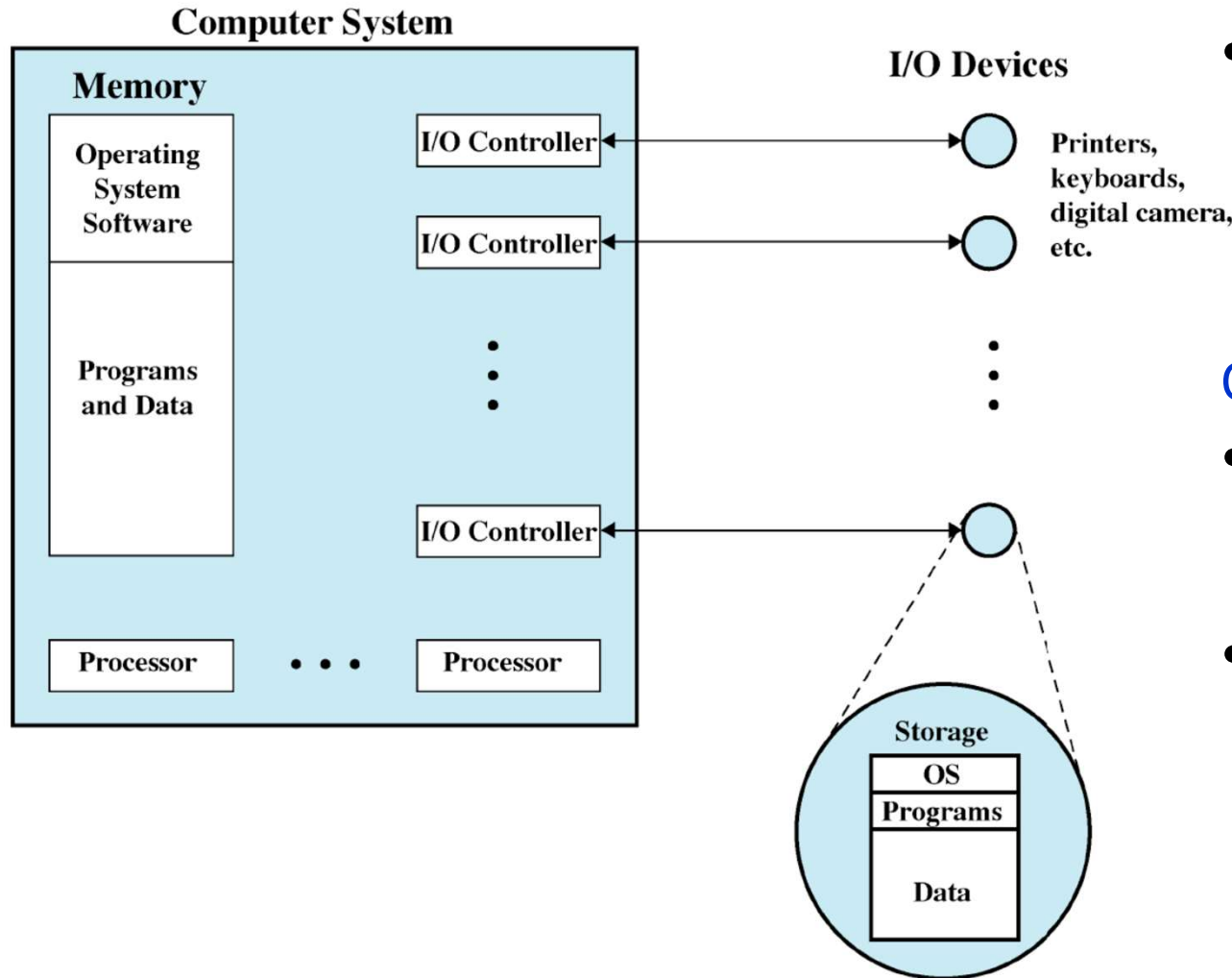


Operating Systems

1. Introduction

Operating System Services

Operating System: ... It is a Program ...



- Relinquishes control of the processor to execute other programs

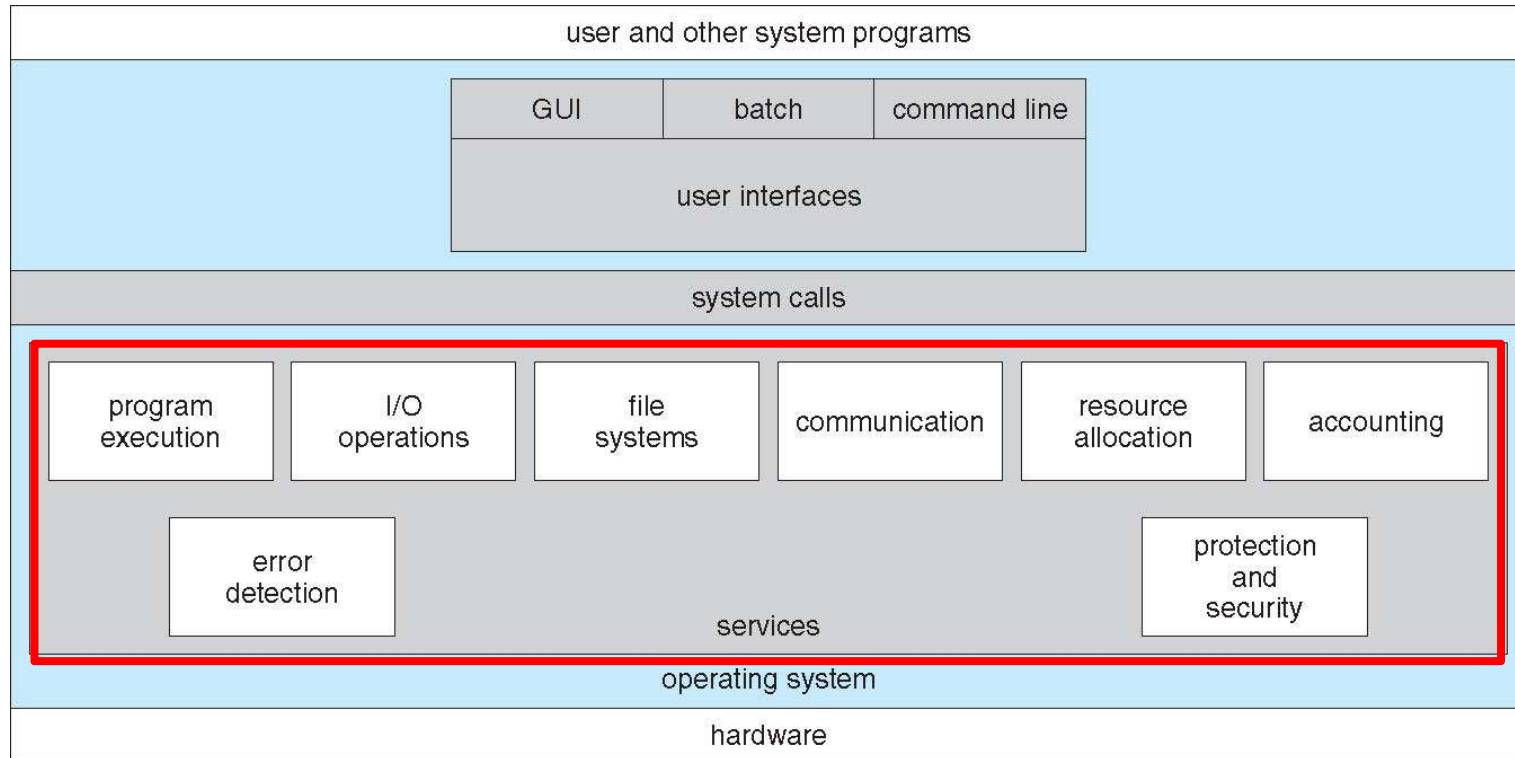
OS Kernel

- (Roughly) Portion of OS that is in main memory
- Contains most-frequently used functions

Operating System in Action

- OS is a program, just like any other program
- On computer start, bootstrap program is loaded from ROM
- Boot program activates OS kernel (permanent system process)
 - Shell (is not kernel): Program to let the user initiate processes
- Boot program
 - Examine/check machine configuration, e.g.,
 - Number of CPUs
 - How much memory
 - Number and type of HW devices
 - Build configuration structure describing the HW
 - Locates and Loads the OS
 - The control transfers to the OS

Operating System Services



Operating System Services – Helper

- User interface

- Command line

- Shell provides command line interface

```
cat file1 file2 file3 | sort > text.txt &  
Create two processes cat, sort and links input/output
```

- Graphical user interface

- Batch interface

- Commands are entered into files and those files are executed

- Program execution

- Load program into memory and run it

- End execution either normally or abnormally

Operating System Services – Helper

- I/O operations
 - User programs cannot control I/O device directly
 - Operating system must provide some means to control I/O
- File system manipulation
 - Program's capability to read, write, create or delete files or directories
 - Permission management to allow or deny access to files or directories
- Communication
 - Exchange of information between processes
 - Processes may be executing on different computers via network
 - Communication on same computer via shared memory or message passing

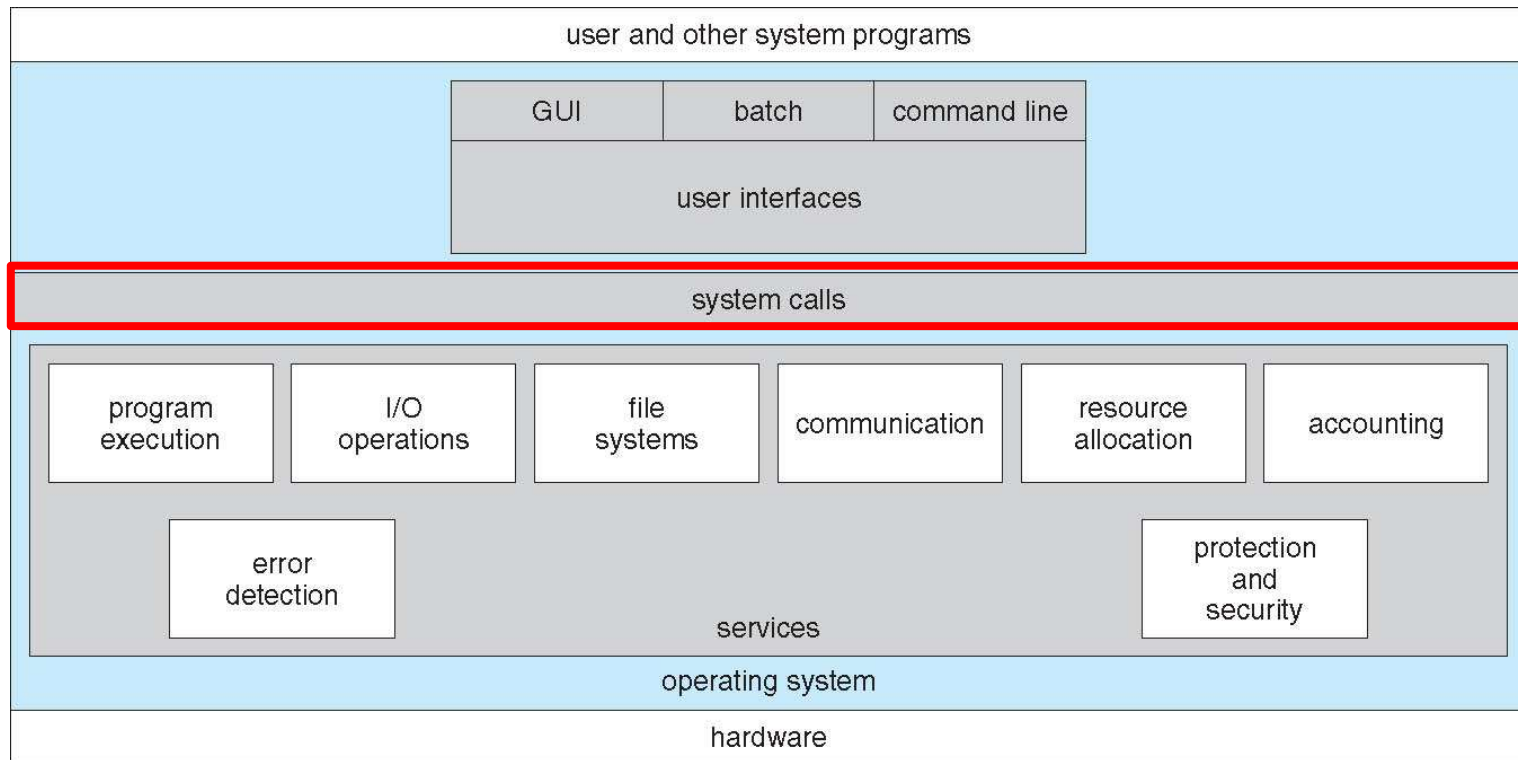
Operating System Services – Helper

- Error detection
 - OS needs to be constantly aware of possible errors
 - May occur in CPU & memory hardware, in I/O device, in user program
 - OS takes corrective actions to ensure correct computing
 - Debugging facilities to the application programmers

Operating System Services – Efficient Operations

- Resource allocation
 - Resource allocation to concurrently running jobs
 - Resources such as CPU cycles, main memory, file storage etc.
- Accounting
 - Track usage of computer resources
 - For billing or simply for accumulating usage statistics
- Protection and security
 - Protect access to system resources is controlled
 - Security from the outsiders, e.g., user authentication

System Calls

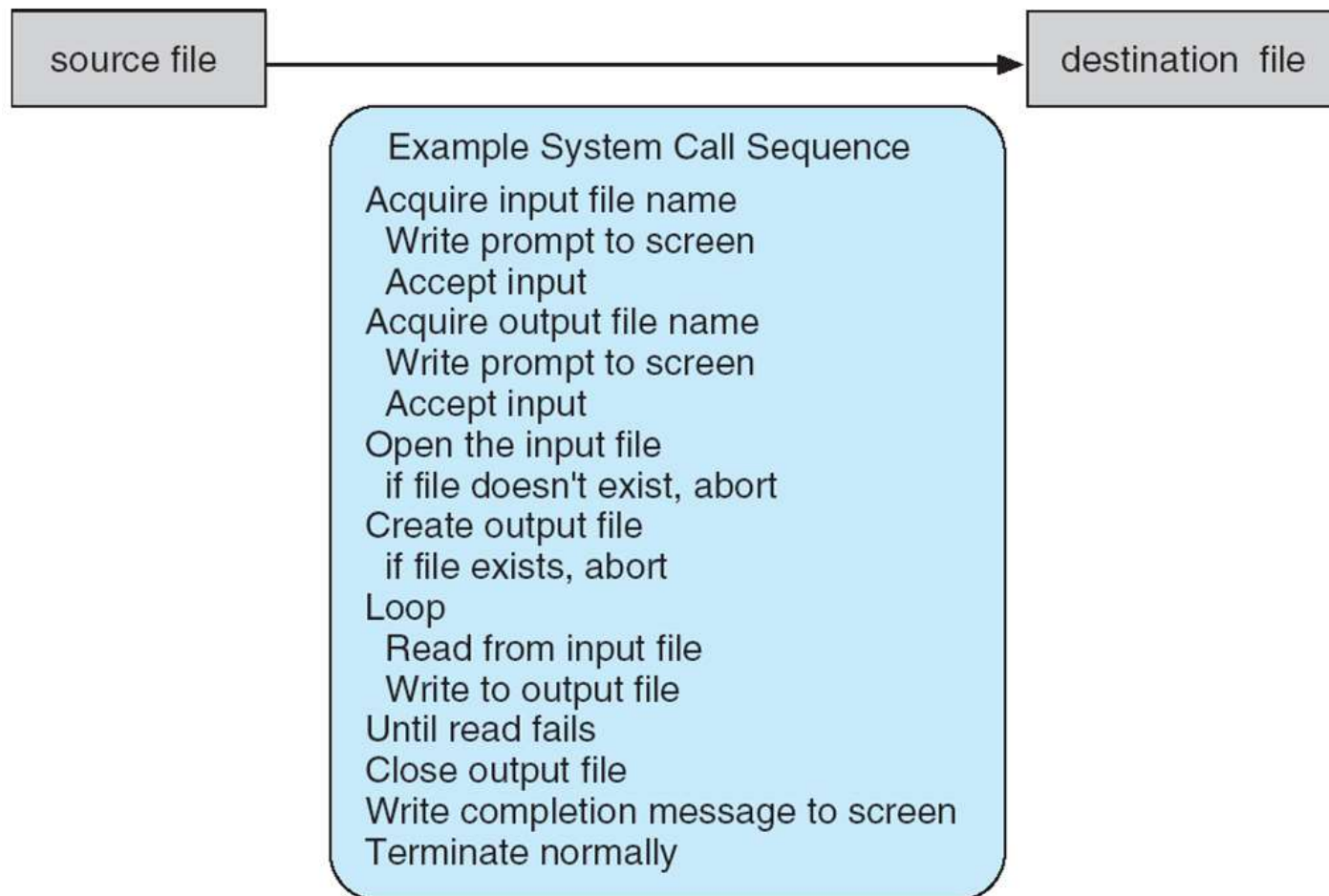


System Calls

- Programming interface to the services provided by the 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
 - [Win32](#) API for Windows
 - [POSIX API](#) for POSIX-based systems
 - Virtually all versions of UNIX, Linux, and Mac OS (X)
 - [Java API](#) for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

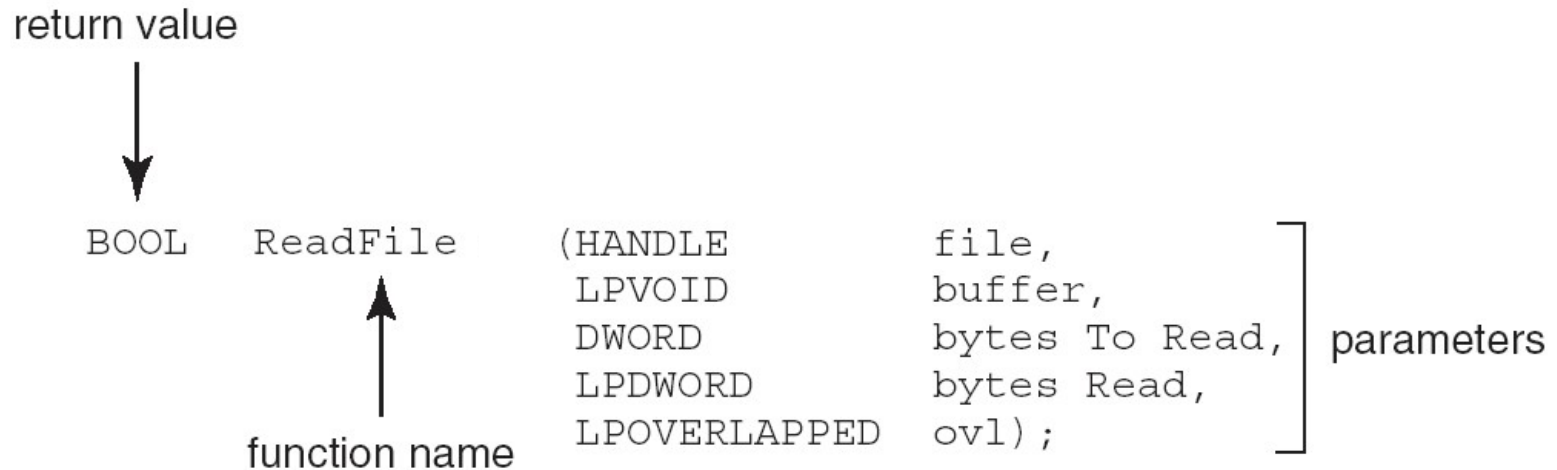
System Call – Example

- Copy the contents of one file to another file



Win32 API Example

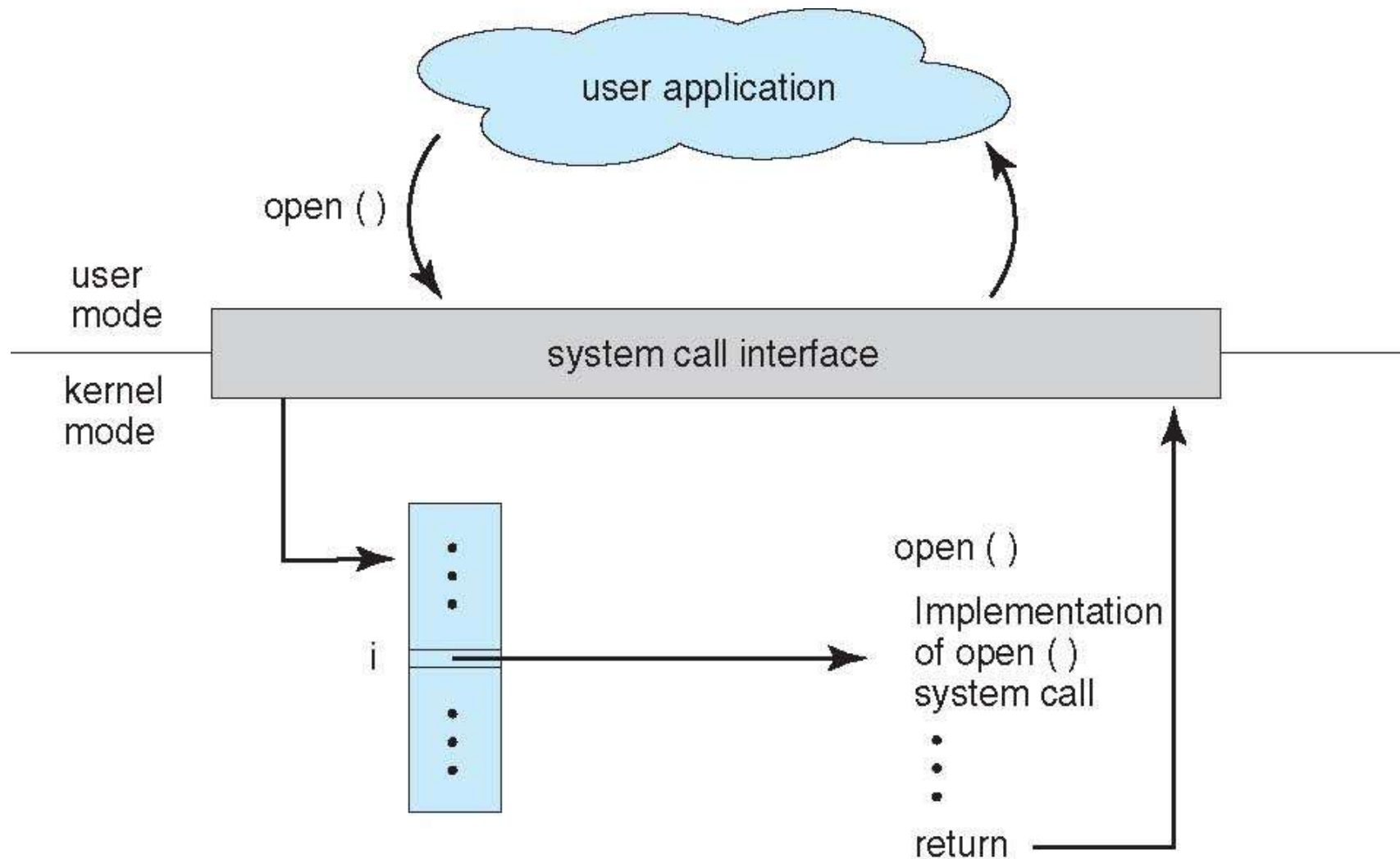
- Consider the `ReadFile()` function in the Win32 API



A description of the parameters passed to `ReadFile()`

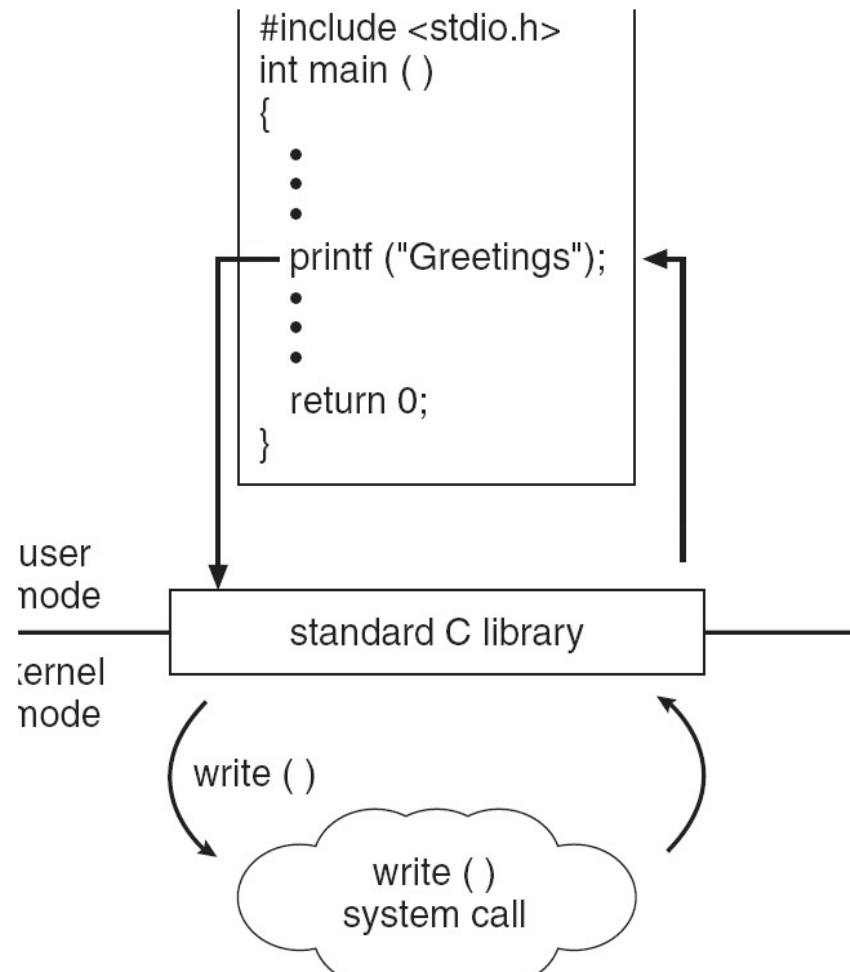
- `HANDLE file`: File to be read
- `LPVOID buffer`: Where the data will be read into and written from
- `DWORD bytesToRead`: Number of bytes to be read into the buffer
- `LPDWORD bytesRead`: Number of bytes read during the last read
- `LPOVERLAPPED ovl`: Indicates if overlapped I/O is being used

API and System Call Relationship

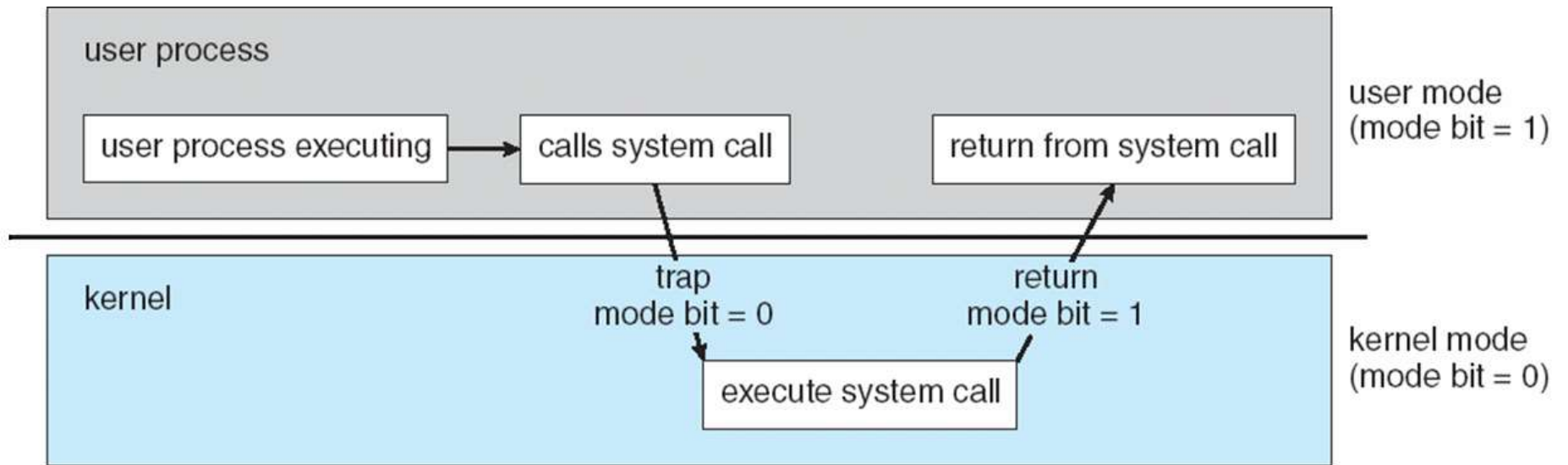


Standard C Library Example

- Program invoke `printf()` library function, which calls `write()` system call



Transition From User to Kernel Mode

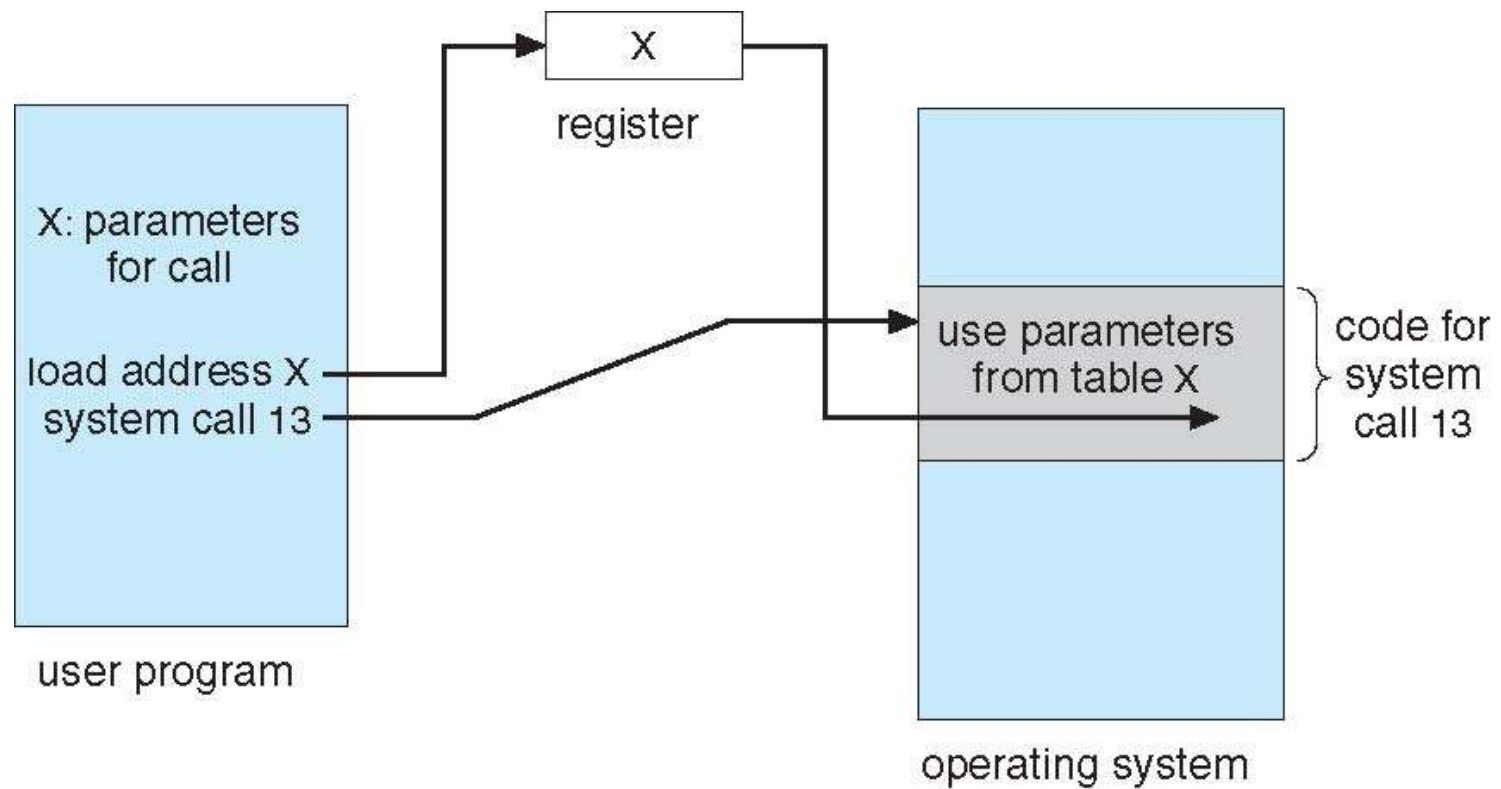


System Call – Parameter Passing

Three general methods used to pass parameters to the OS

- 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



Types of System Calls

- **Process control**
 - End, abort
 - Load, execute
 - Create process, terminate process
 - Get process attributes, set process attributes
 - Wait for time
 - Wait event, signal event
 - Allocate and free memory
- **File management**
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes

Types of System Calls

- **Device management**
 - Request device, release device
 - Read, write, reposition
 - Get device attributes, set device attributes
 - Logically attach or detach devices
- **Information maintenance**
 - Get time or date, set time or date
 - Get system data, set system data
 - Get and set process, file, or device attributes
- **Communications**
 - Create, delete communication connection
 - Send, receive messages
 - Transfer status information
 - Attach and detach remote devices

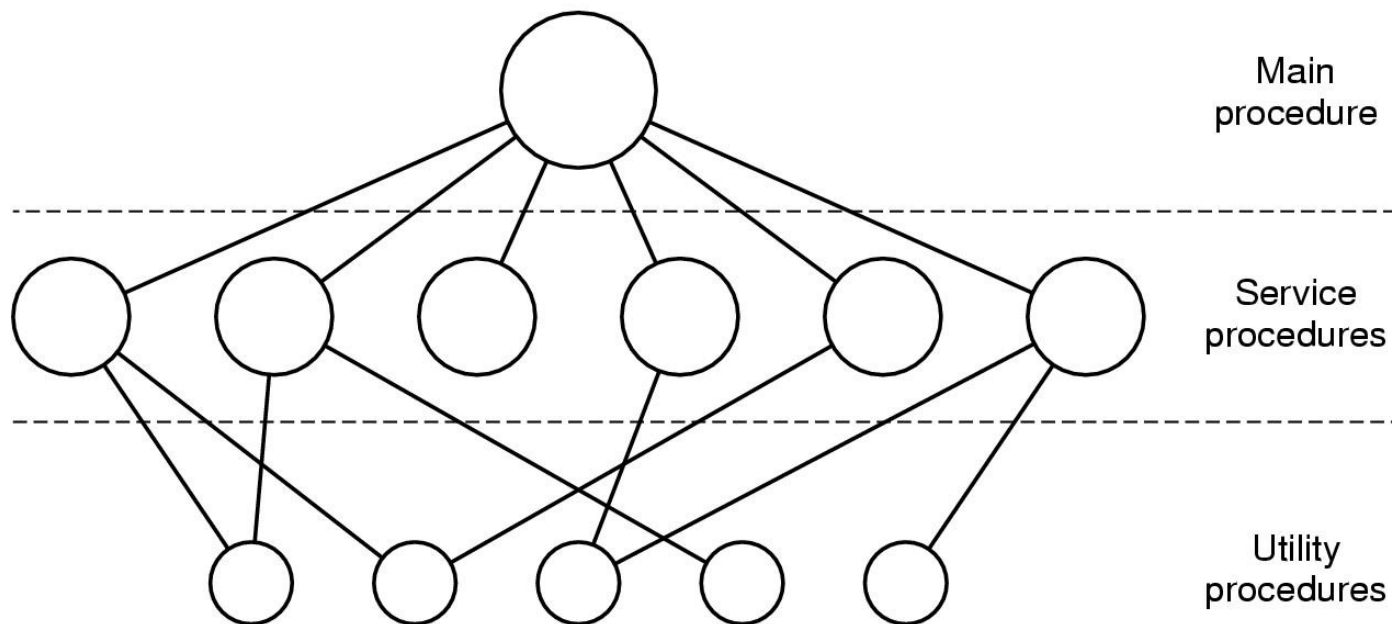
Windows and Unix System Calls – Example

	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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Operating System Structure

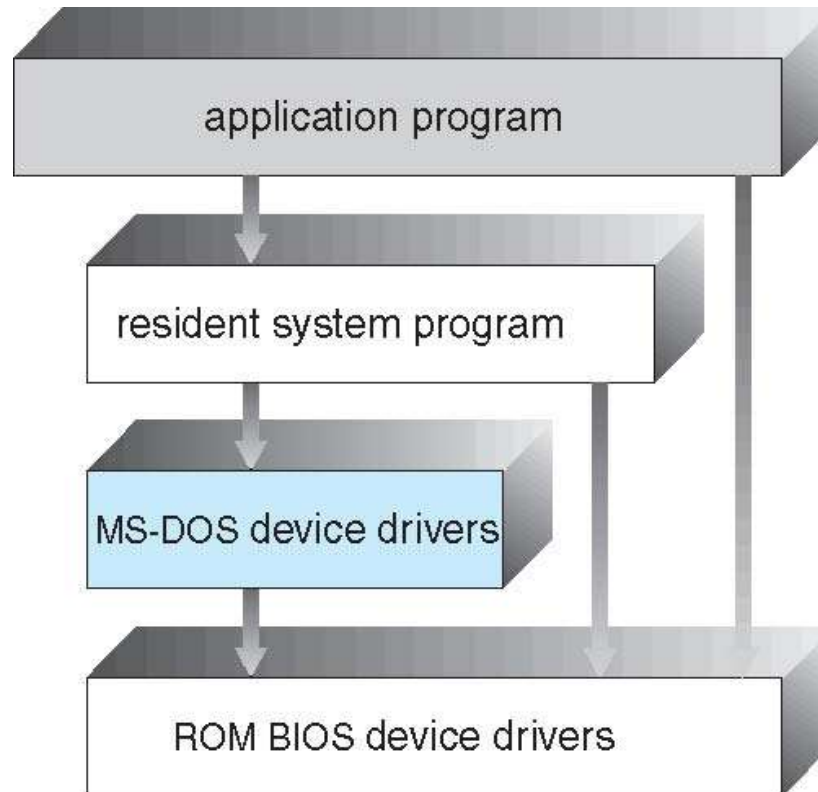
Monolithic Systems

- Most operating systems have relied on a monolithic structure
 - OS is written as a collection of procedures
 - Each procedure can call any other procedure
 - Issues: difficult to implement and maintain



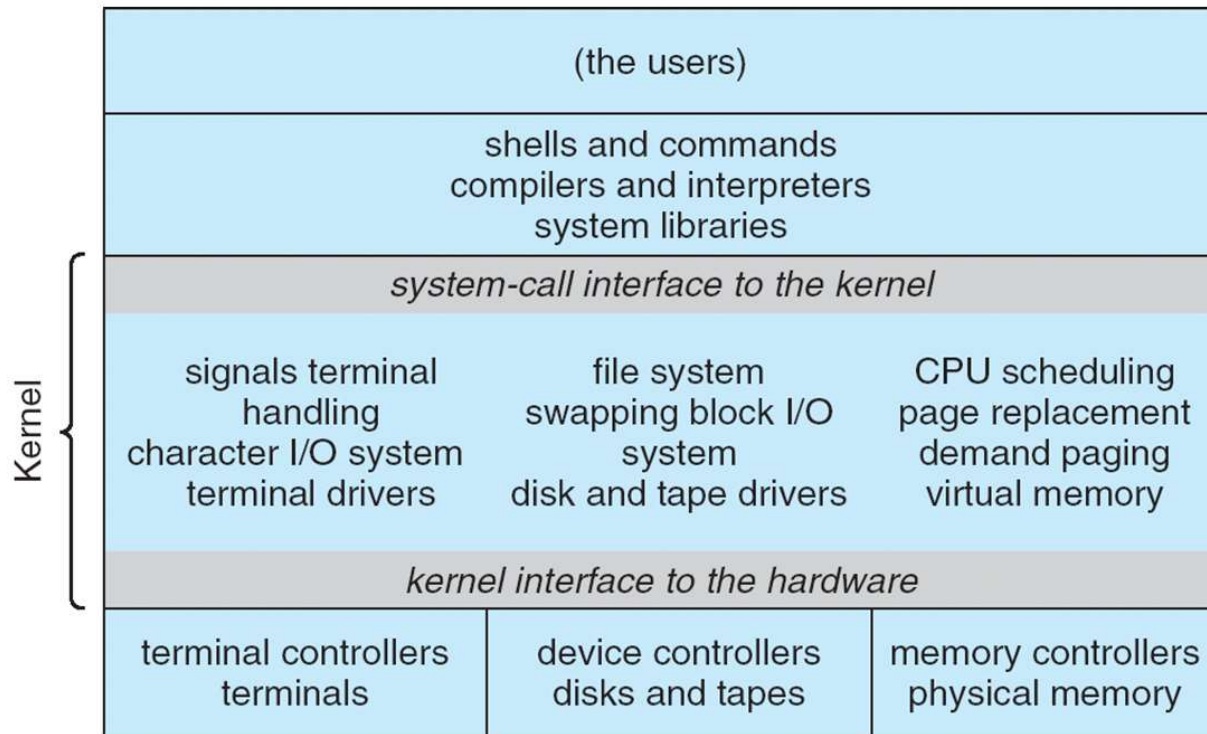
MS-DOS

- Written to provide the most functionality in the least space
 - Not divided into modules
 - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



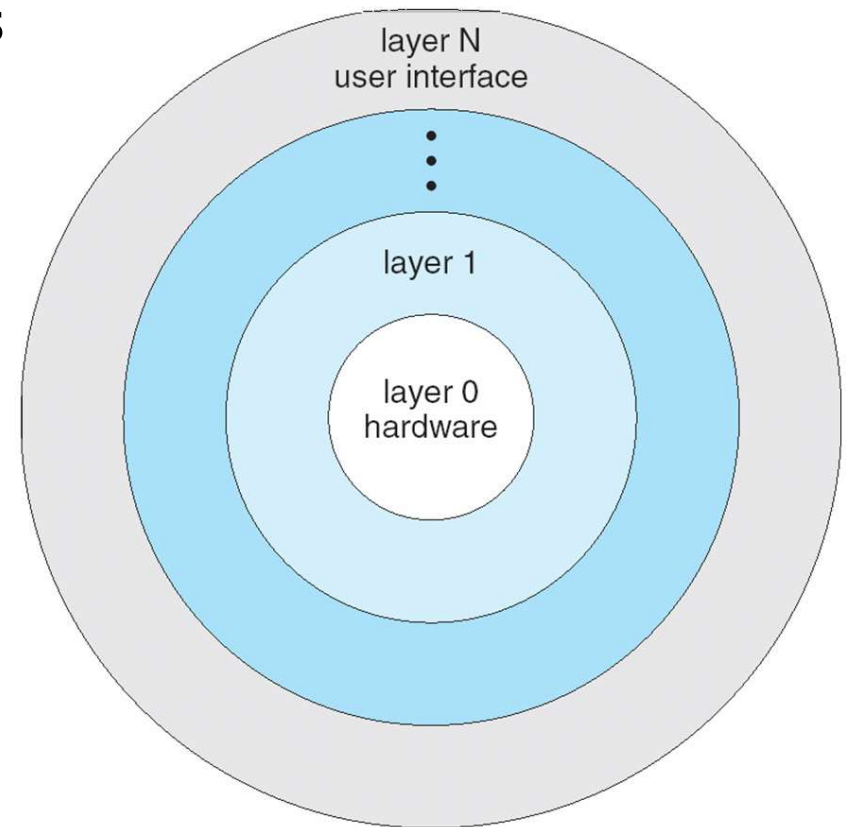
UNIX

- Traditional UNIX OS consists of two separable parts
 - Systems programs
 - Kernel
- Kernel consists of everything below the system-call interface and above the physical hardware

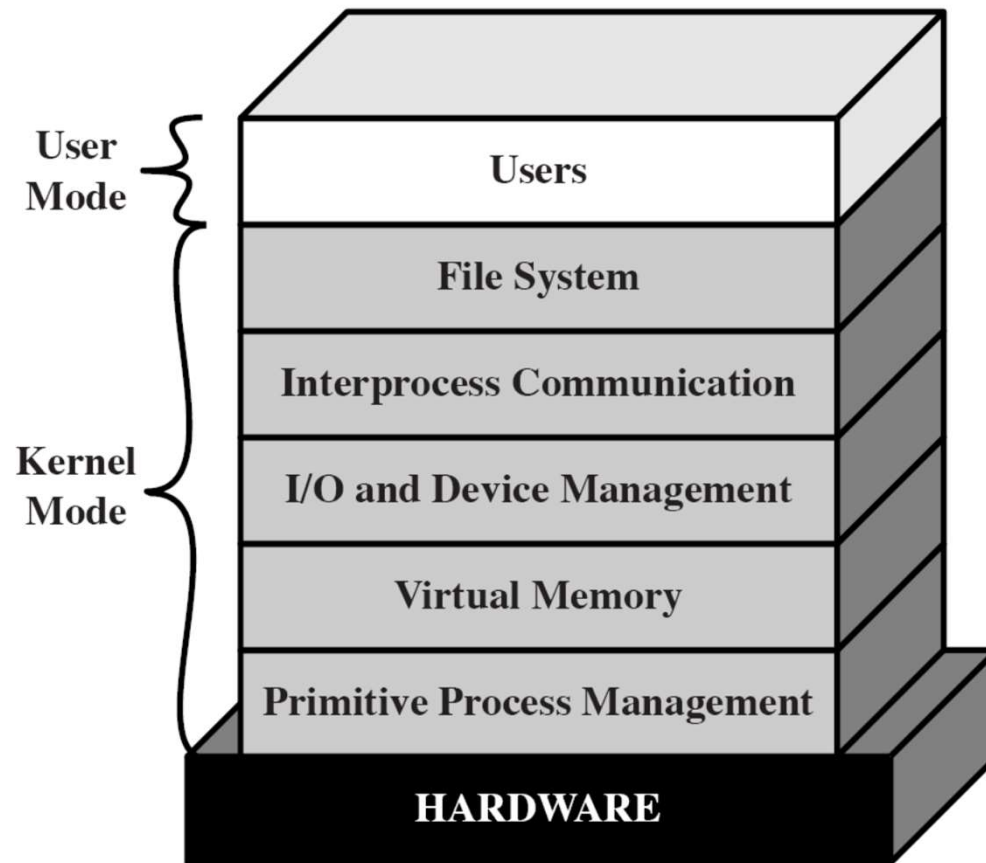


Layered Approach

- Break operating system into pieces
- Each layer interacts only with its lower layer
- Benefits
 - Allows to build modular systems
 - Can exchange each layer
 - Facilitates testing
 - Look at each layer separately
- Drawback
 - Difficult to define appropriate layers
 - Sometimes contradictory needs
 - Tends to be less efficient

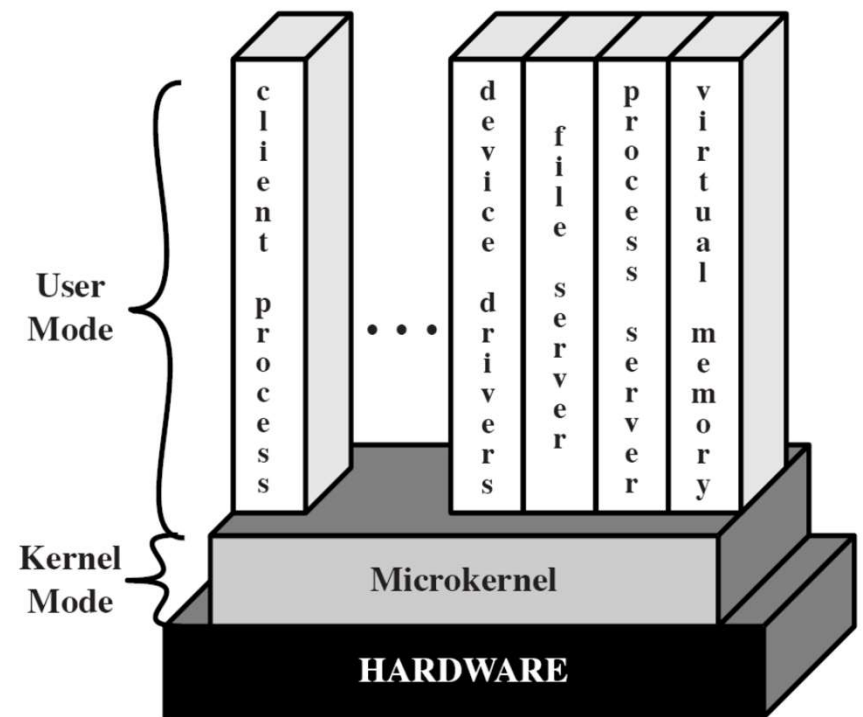


Layered Approach – Example



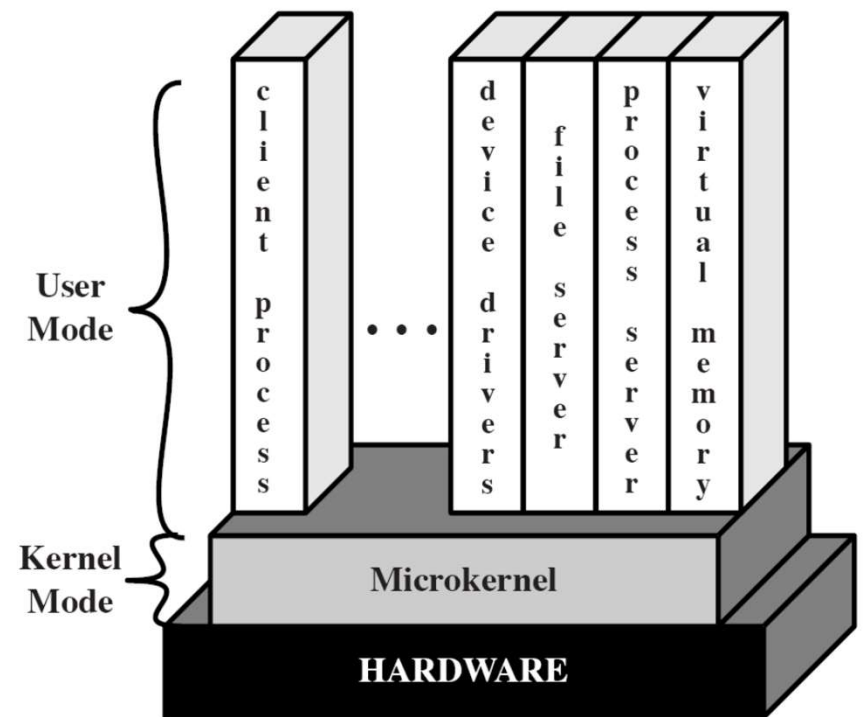
Microkernel

- Small OS core – contains only essential OS functions
 - Low-level memory management
 - Process scheduling
 - Communication facility
- Many services traditionally included in the OS kernel are now external subsystems
 - Device drivers, file systems, virtual memory manager, windowing system, security services
 - Run in user mode

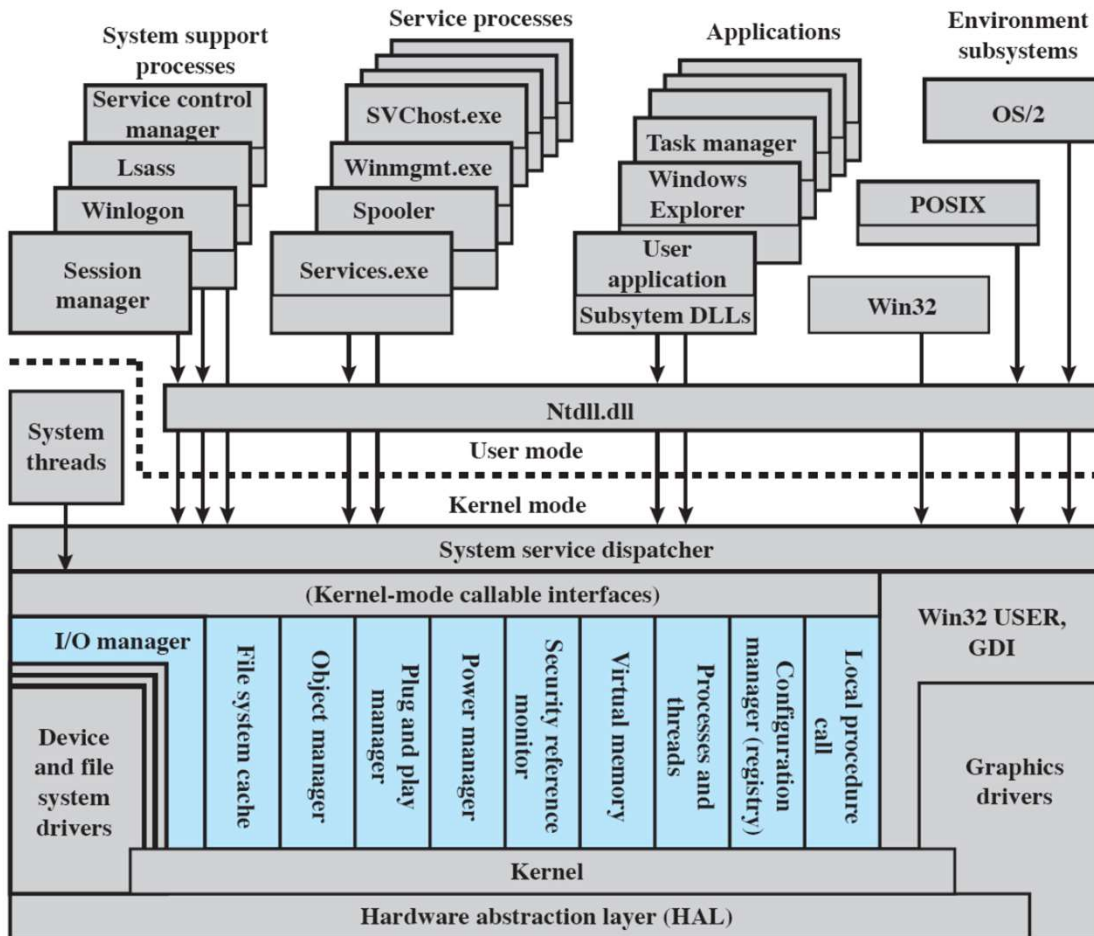


Microkernel

- Corresponds to client/server model
 - Communication takes place between user modules using message passing
- Benefits
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments
 - Performance overhead of user space to kernel space communication



Windows 2000



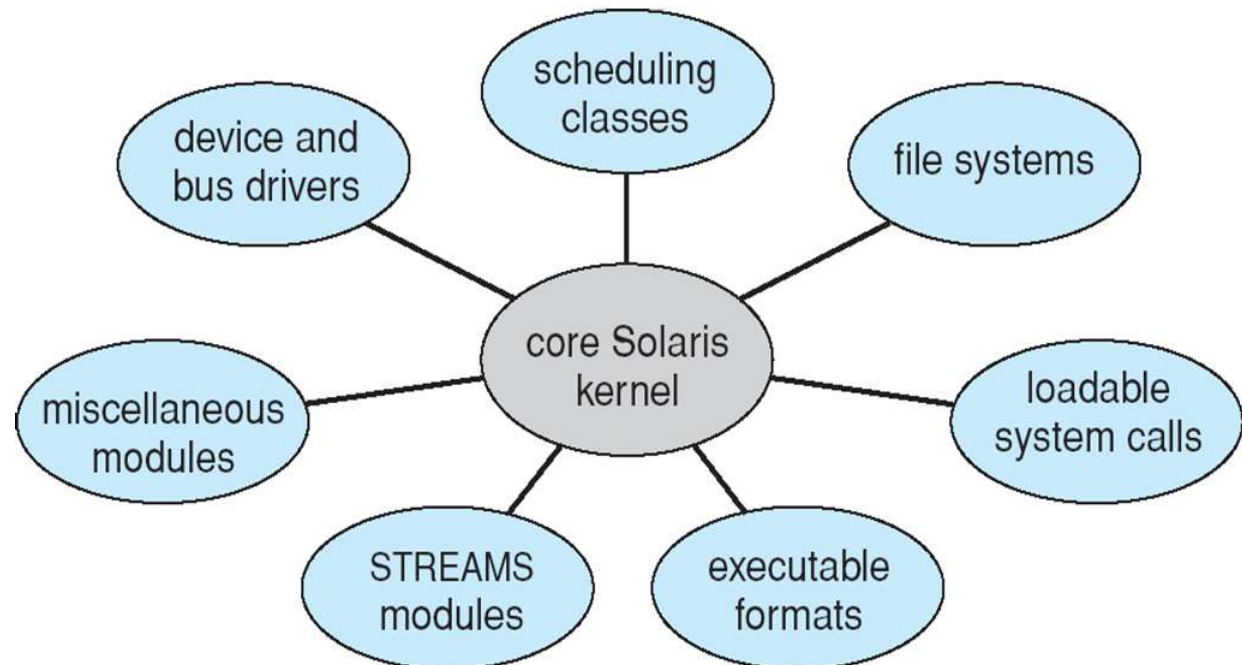
Lsass = local security authentication server
 POSIX = portable operating system interface
 GDI = graphics device interface
 DLL = dynamic link libraries

Colored area indicates Executive

- Client/Server computing
- Modified **microkernel** architecture
 - Not a pure microkernel: many system functions outside of the microkernel run in kernel mode
 - Modules can be removed, upgraded, or replaced without rewriting the entire system

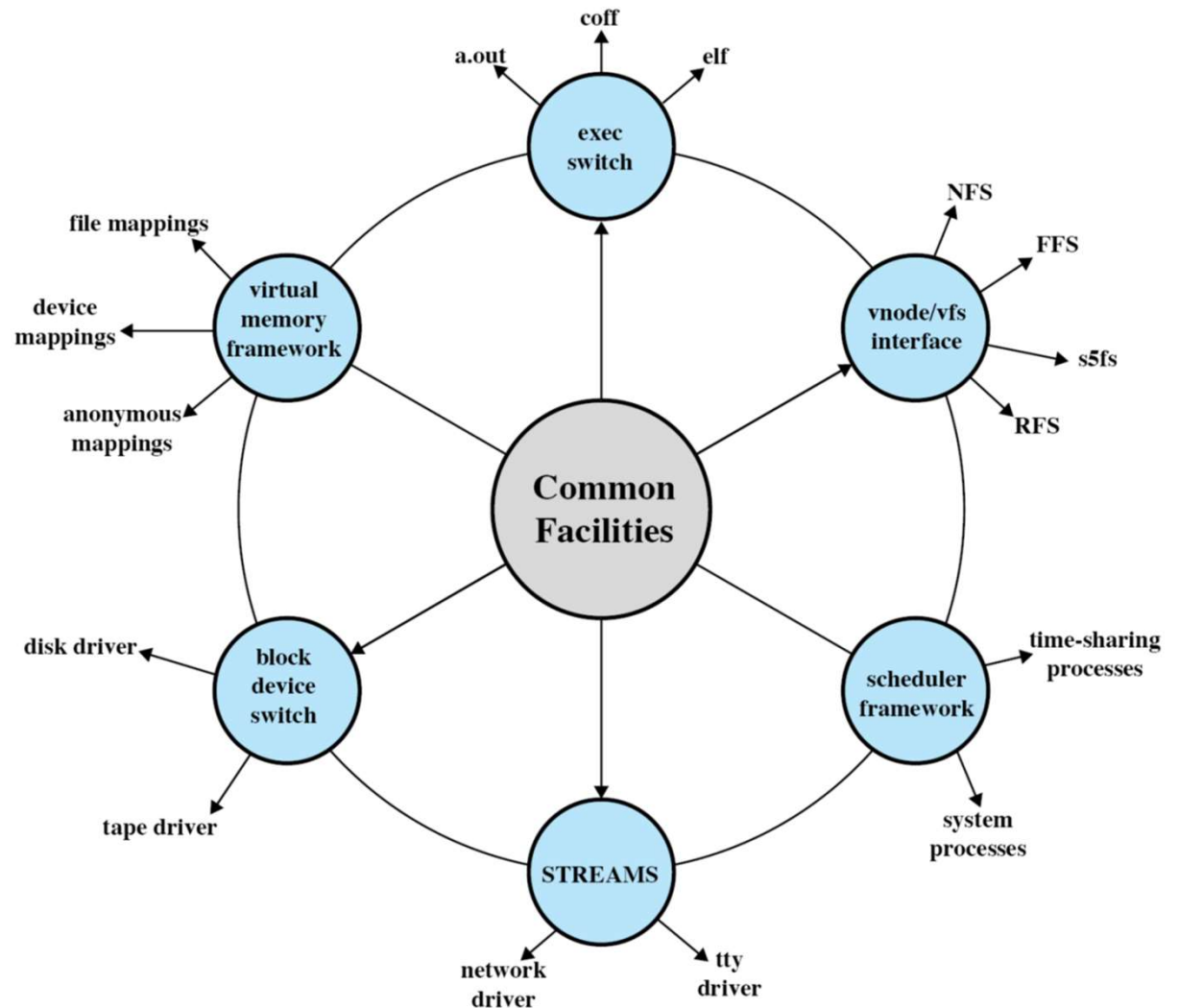
Modules

- Most modern operating systems implement kernel modules
 - 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 with more flexible



Modular Approach in Modern UNIX Instances

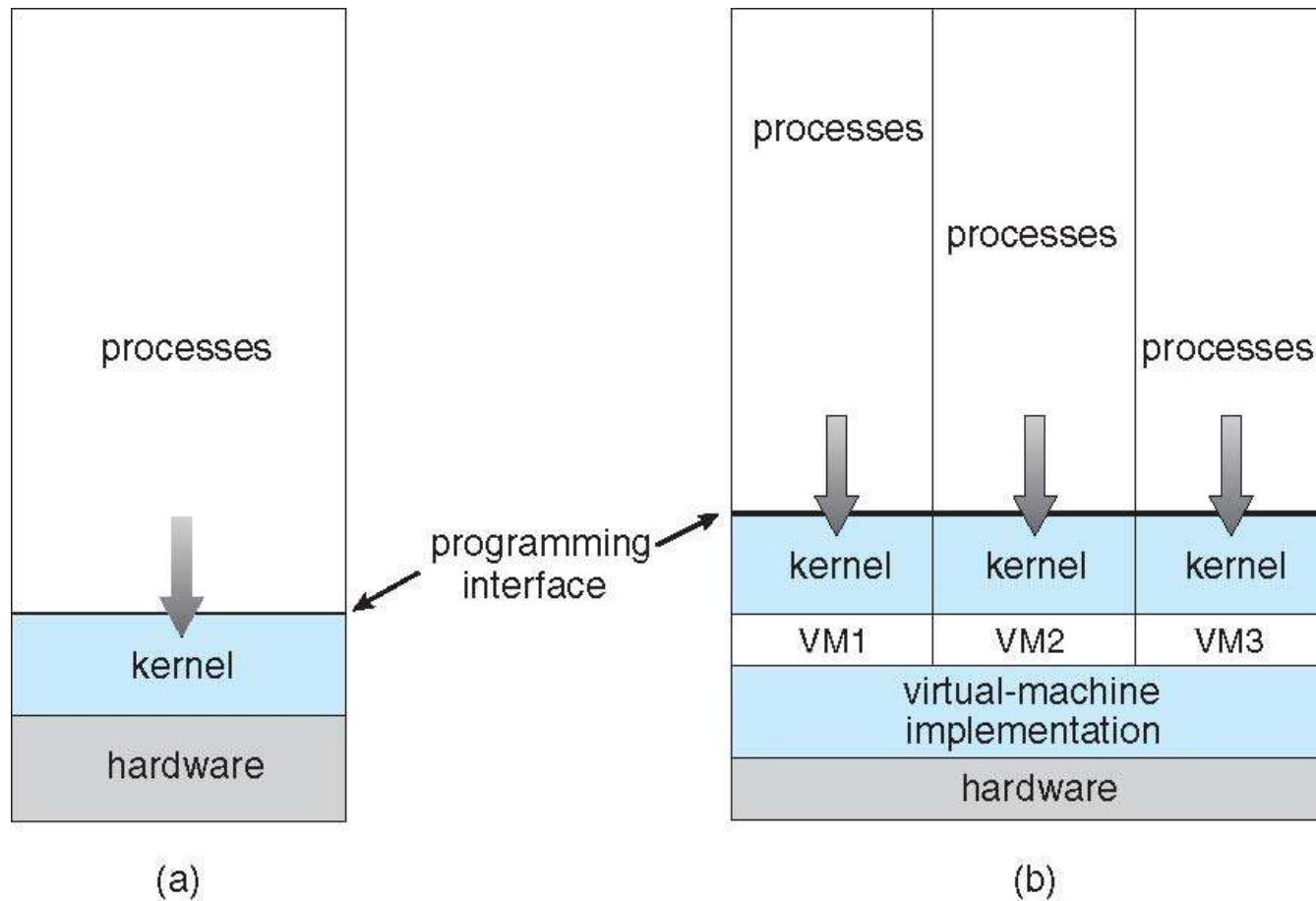
- **Kernel** set of core facilities
- **Modules** can be dynamically linked during run time
- Modern Unix approaches, e.g., Solaris follow a modular organization



Virtual Machines

- Abstract the hardware of a single computer into several different execution environment
 - CPU, memory, disk drives, network interface cards and so forth
- Creates the illusion that each execution environment is running its own private computer
- A virtual machine provides interface identical to underlying bare hardware
 - I.e., all devices, interrupts, memory, page tables, etc.
- Virtual Machine Operating System creates illusion of multiple processors
 - Each capable of executing independently
 - No sharing, except via network protocols

Virtual Machines

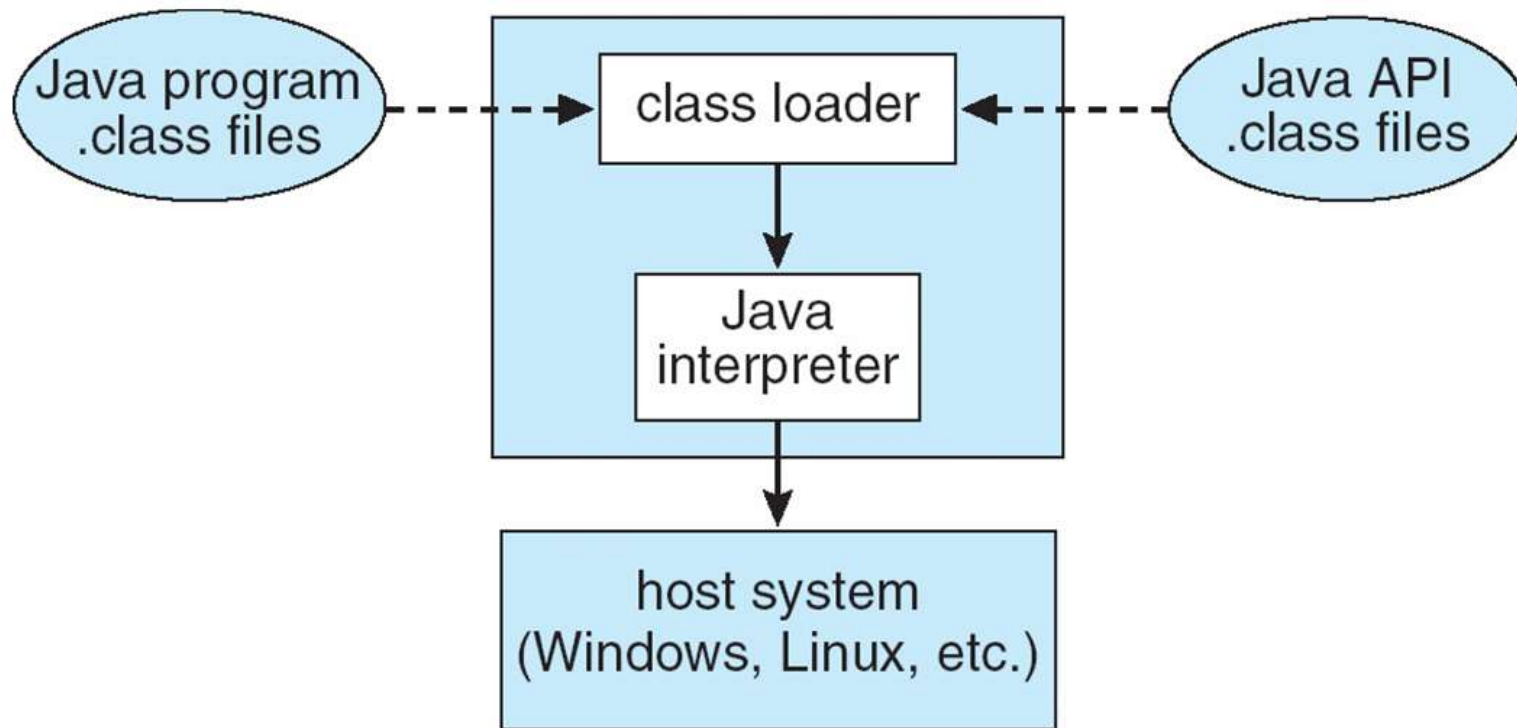


(a) Non-virtual machine (b) Virtual machine

Java Virtual Machine

- Compiled Java programs are platform-neutral byte-codes
 - Executed by a Java Virtual Machine (JVM)
- JVM consists of
 - Class loader
 - Class verifier
 - Runtime interpreter
- Class verifier checks
 - .class file is valid Java byte code
 - Does not overflow or underflow the stack
 - Byte code does not perform illegal memory access
- Just-In-Time (JIT) compilers increase performance

Java Virtual Machine



Any Question So Far?

