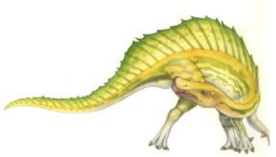
CSE303 Fundamentals of Operating Systems

Assoc.Prof.Dr. Taner DANISMAN

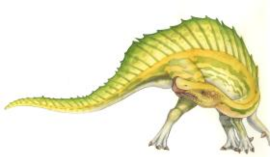
tdanisman@akdeniz.edu.tr



Syllabus

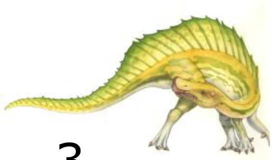
- Semester: Fall 2025
- Instructor : Taner Danisman

- **Lectures**
  - 40 minutes course + 15 minutes break
  - Location:
    - Tuesday :15:30-16:20 (Amfi 4)
    - Thursday :10:30-12:20 (D206)
- **TextBook:**
  - Operating System Concepts, 9th edition, Silberschatz et al. Wiley, 2013
  - Modern Operating Systems, Andrew S. Tanenbaum, 4th Edition, Pearson, 2015



Prerequisites

- Readers familiarity with
  - Basic data structures
  - Computer organization
  - C programming
- **Reading:**
  - C Coding
    - https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html

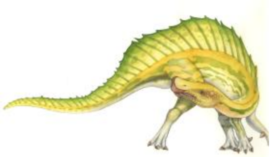


Lab material

- OS: Ubuntu 20.04.* LTS or higher (Virtual or installed OS)
  - MINIX Operating System
    - DOSMINIX 2.0.4 (Book edition)
      - https://minix1.woodhull.com/current/2.0.4/
  - gcc compiler
- VirtualBox 6.1+ (Oracle)

Grading

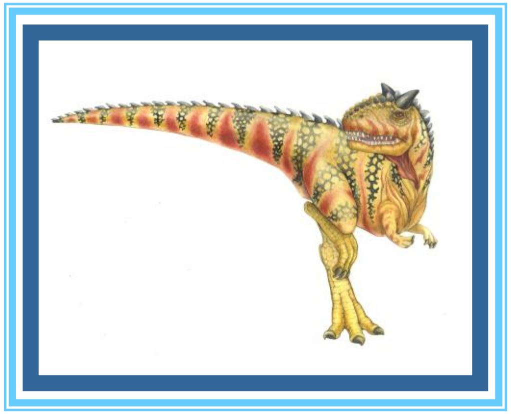
- Absolute grading
- Assignments : 20% (4 assignments, different weights)
  - Group assignments is possible
- Midterm:30%
- Final 50%



Topics to be Covered

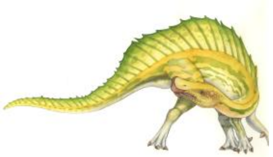
Week 1	Introduction
Week 2	Operating System Structures
Week 3	Processes
Week 4	Threads
Week 5	Threads
Week 6	Synchronization
Week 7	Synchronization
Week 8	Midterm
Week 9	Classical Problems
Week 10	CPU Scheduling
Week 11	CPU Scheduling
Week 12	Deadlocks
Week 13	Memory Management
Week 14	Virtual Memory

Chapter 1: Introduction

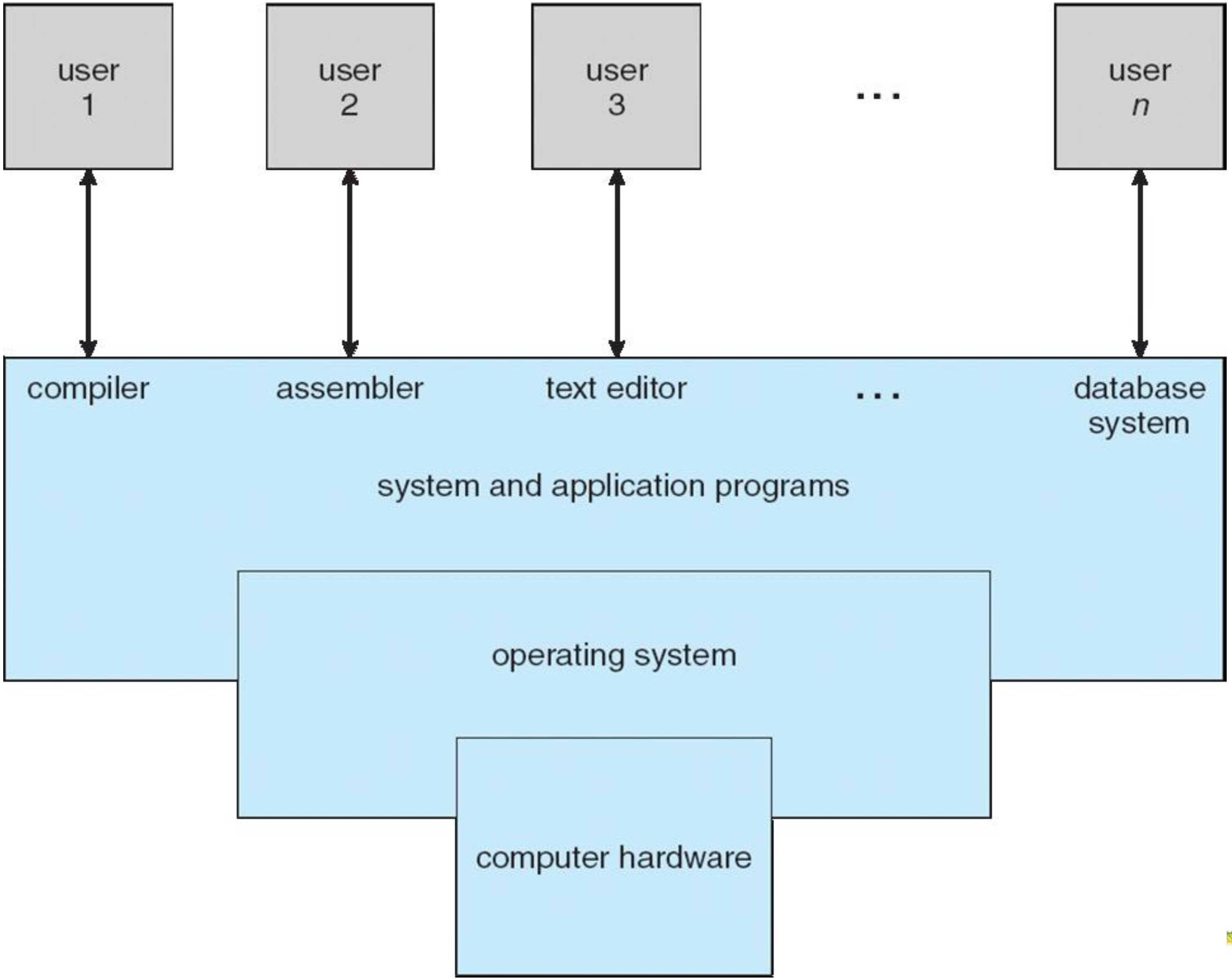


What is an Operating System?

- A program that acts as an **intermediary** between a user of a computer and the computer hardware.
- An operating system is software that manages the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system **convenient** to use
  - Use the computer **hardware** and **resources** in an **efficient** manner



Four Components of a Computer System

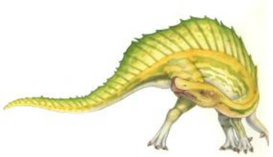


What Operating Systems Do – User View

- Depends on the point of view
- Users want **convenience**, **ease of use** and **good performance**
  - Don’t care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users (terminals) happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, **optimized for usability** and **battery life**
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

Operating System Definition

- From the computer’s point of view, the operating system is the program mostly involved with the hardware.
- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and **fair resource use**
- OS is a **control program**
  - Controls execution of programs to prevent errors and **improper use** of the computer

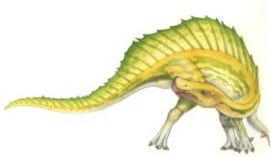


Operating System Definition (Cont.)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
  - Microsoft case in 1998
- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a system program (ships with the operating system) , or
  - an application program.

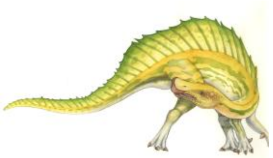
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**
  - When a system is fully booted, it waits for some event to occur
  - The occurrence of an event is usually signaled by an **interrupt**
    - It can be a **hardware** or **software** interrupt
      - **Hardware** interrupts**: Sending signal to the CPU**
      - **Software** interrupts**: Systems calls**



Interrupt Handling

- When the CPU is interrupted:
  - The interrupt must transfer control to the appropriate interrupt service routine
  - it stops what it is doing and immediately transfers execution to a fixed location
  - The fixed location usually contains the **starting address** where the service routine for the interrupt is located
  - The interrupt service routine executes; on completion, the CPU resumes the interrupted computation

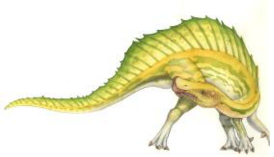


Interrupt Handling (cont.)

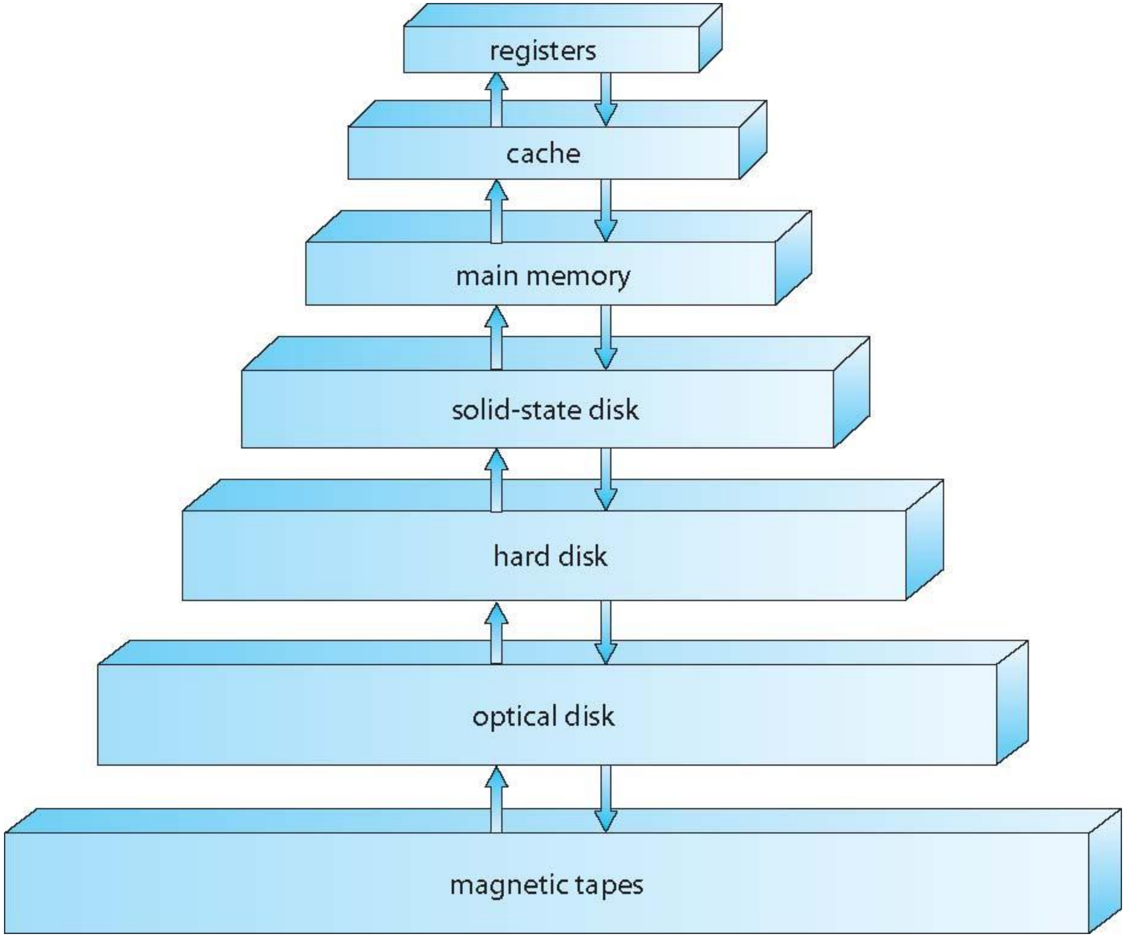
- The operating system preserves the state of the CPU by storing **registers** and the **program counter**
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt
- Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. Windows and UNIX dispatch interrupts in this manner
- After the interrupt is serviced, the saved return address is loaded into the program counter, and the interrupted computation resumes as though the interrupt had not occurred.

Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer
- **Solid-state disks** – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

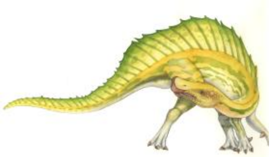


Storage-Device Hierarchy



Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- **Only one interrupt is generated per block**, rather than the one interrupt per byte

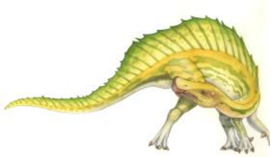


Computer-System Architecture

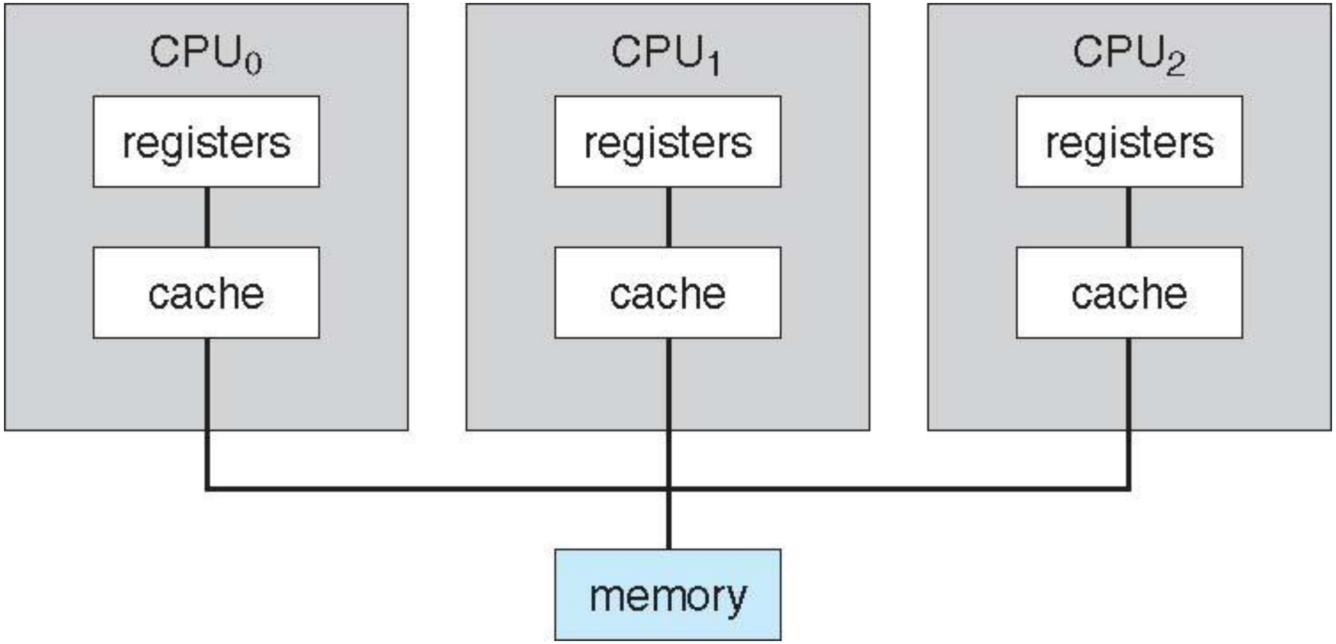
- Old systems use a **single general-purpose processor**
  - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale,** shared resources, memory etc.
    3. **Increased reliability** – **graceful degradation** or **fault tolerance**
       - The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**
       - **fault tolerant** systems can suffer a failure of any single component and still continue operation
         - The HP NonStop (formerly Tandem) system uses both hardware and software duplication to ensure continued operation despite faults

Computer-System Architecture

- Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
        1. A **boss** processor controls the system; boss-worker relationship
    2. **Symmetric Multiprocessing (SMP)** – each processor performs all tasks
        1. no boss–worker relationship exists
- Multiprocessing can cause a system to change its memory access model from **uniform memory access (UMA)** to **non-uniform memory access (NUMA)**
- **UMA** is defined as the situation in which access to any RAM from any CPU takes the same amount of time.
- With **NUMA**, some parts of memory may take longer to access than other parts, creating a performance penalty.
- **multicore systems are multiprocessor systems**

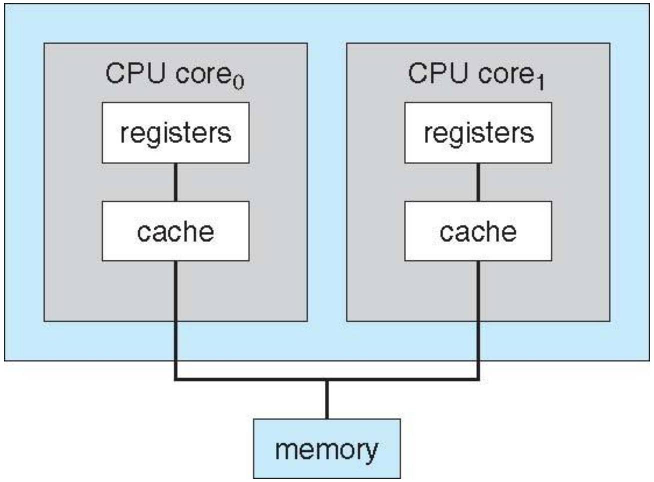


Symmetric Multiprocessing Architecture



A Dual-Core Design

- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems

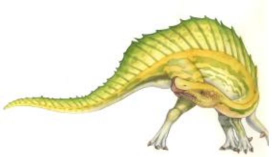


Operating-System Operations

- **Interrupt driven** (hardware and software)
    - Hardware interrupt by one of the devices
    - Software interrupt (**exception** or **trap):**
        - Software error (e.g., division by zero)
        - Request for operating system service
        - Other process problems include
            - **infinite loop**
            - Processes modifying each other or the operating system

Operating-System Operations (cont.)

- **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



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 reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of 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

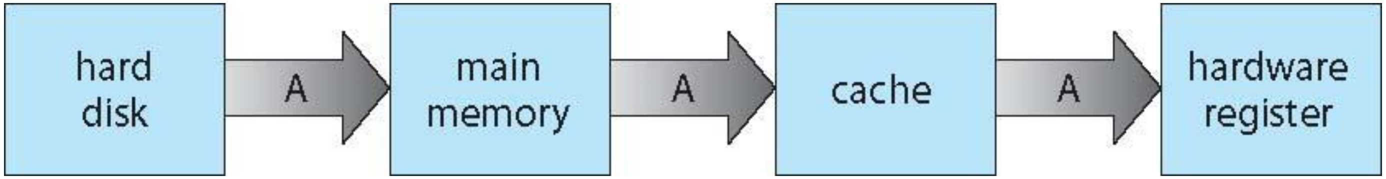
Process Management Activities

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 **synchronization**
- Providing mechanisms for process **communication**
- Providing mechanisms for **deadlock** handling

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

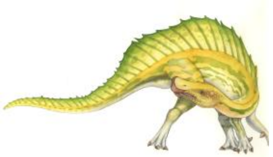


- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17

Performance of Various Levels of Storage

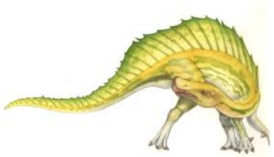
Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 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

Movement between levels of storage hierarchy can be explicit or implicit



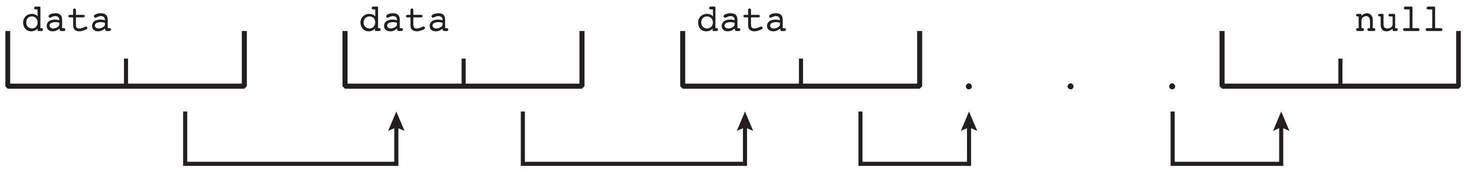
Protection and Security

- **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 effective ID with more rights

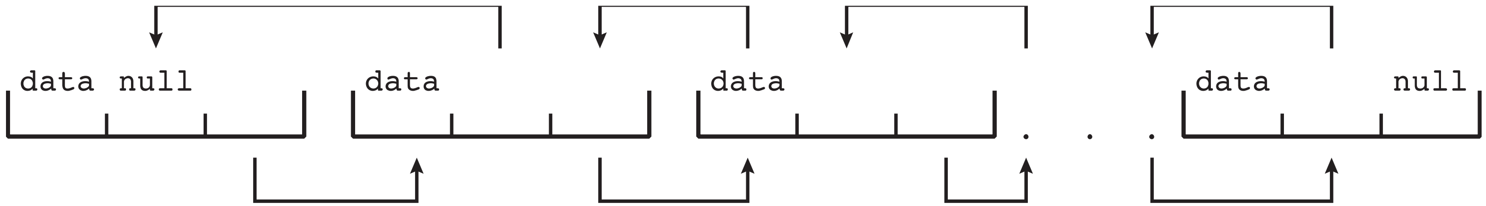


Kernel Data Structures

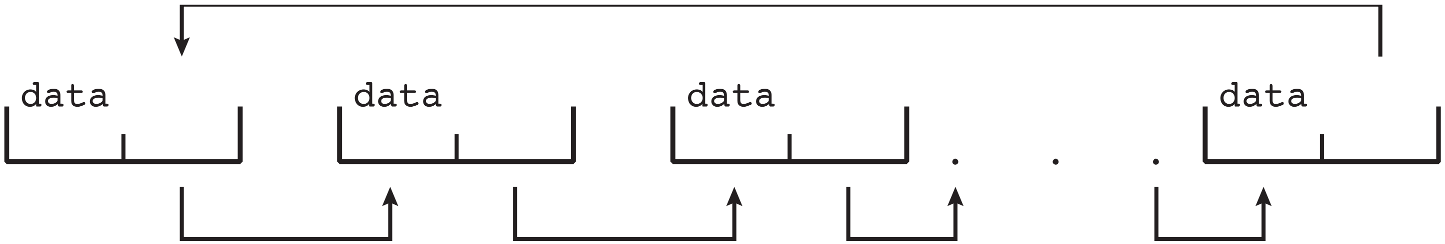
- Many similar to standard programming data structures
- ***Singly linked list***



- ***Doubly linked list***

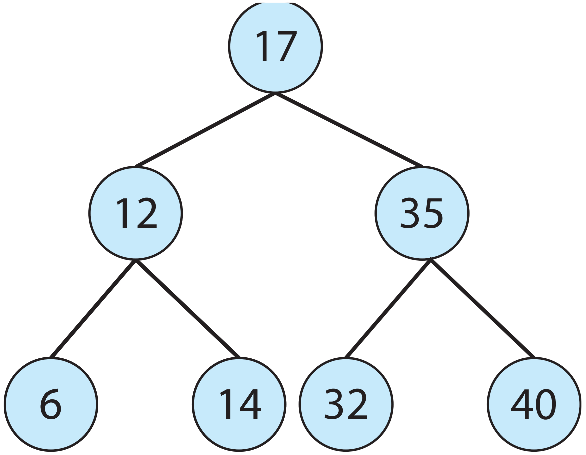


- ***Circular linked list***



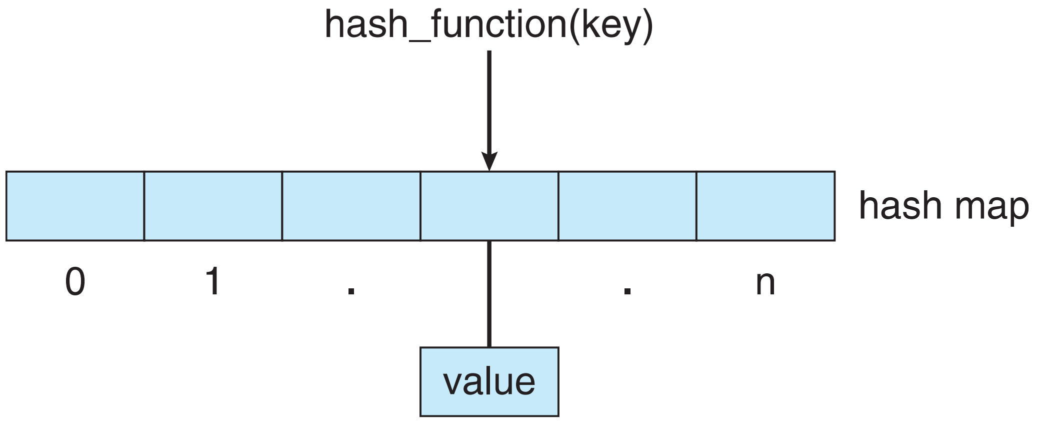
Kernel Data Structures

- **Binary search tree**
  left <= right
  - Search performance is *O(n)*
  - **Balanced binary search tree** is *O(log n)*



Kernel Data Structures

- **Hash function** can create a **hash map**



- **Bitmap** – string of *n* binary digits representing the status of *n* items
- Linux data structures defined in

  include files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Computing Environments – Distributed

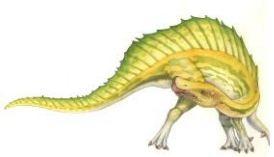
- Distributed computiing
  - Collection of separate, possibly heterogeneous, systems networked together
    - **Network** is a communications path, **TCP/IP** most common
      - **Local Area Network** (**LAN**)
      - **Wide Area Network** (**WAN**)
      - **Metropolitan Area Network** (**MAN**)
      - **Personal Area Network** (**PAN**)
  - **Network Operating System** provides features between systems across network
    - Communication scheme allows systems to exchange messages
    - Illusion of a single system

Computing Environments - Virtualization

- Allows operating systems to run applications within other OSes
  - Vast and growing industry
- **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
  - Generally slowest method
  - When computer language not compiled to native code – **Interpretation**
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
  - Consider VMware running Win10 guests, each running applications, all on native Win10 **host** OS
  - **VMM** (virtual machine Manager) provides virtualization services

Computing Environments – Cloud Computing

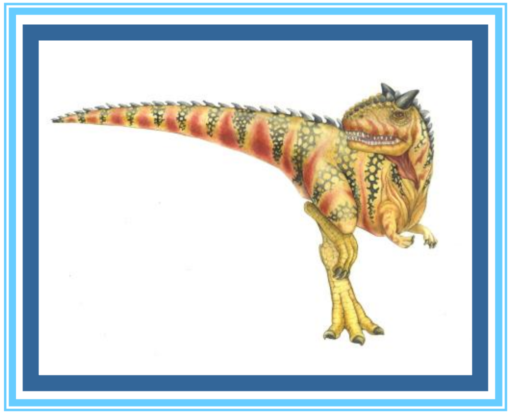
- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **Public cloud** – available via Internet to anyone willing to pay
  - **Private cloud** – run by a company for the company's own use
  - **Hybrid cloud** – includes both public and private cloud components
  - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)



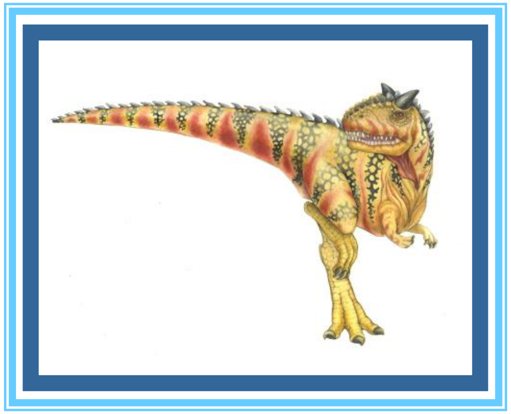
Computing Environments – Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
  - Vary considerable, special purpose, limited purpose OS, **real-time OS**
  - Use expanding
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met

End of Chapter 1



Chapter 2: Operating-System Structures



Schedule

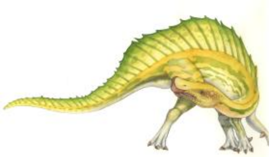
Week 1	Introduction
Week 2	Operating System Structures
Week 3	Processes
Week 4	Threads
Week 5	Threads
Week 6	Midterm 1
Week 7	Synchronization
Week 8	Classical Problems
Week 9	CPU Scheduling
Week 10	CPU Scheduling
Week 11	Midterm 2
Week 12	Deadlocks
Week 13	Memory Management
Week 14	Virtual Memory
Week 15	Virtual Memory

Operating System Services

- Operating systems provide:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - **Command-Line Interface(CLI)**, text commands
    - **Graphics User Interface (GUI)**,
    - **Batch Interface,** commands and directives are entered intoa file and file is executed
  - **Program execution**
  - **I/O operations**
  - **File-system manipulation** Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)

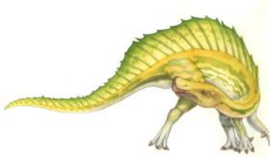
Operating System Services (Cont.)

- (Cont.):
  - **Error detection**
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

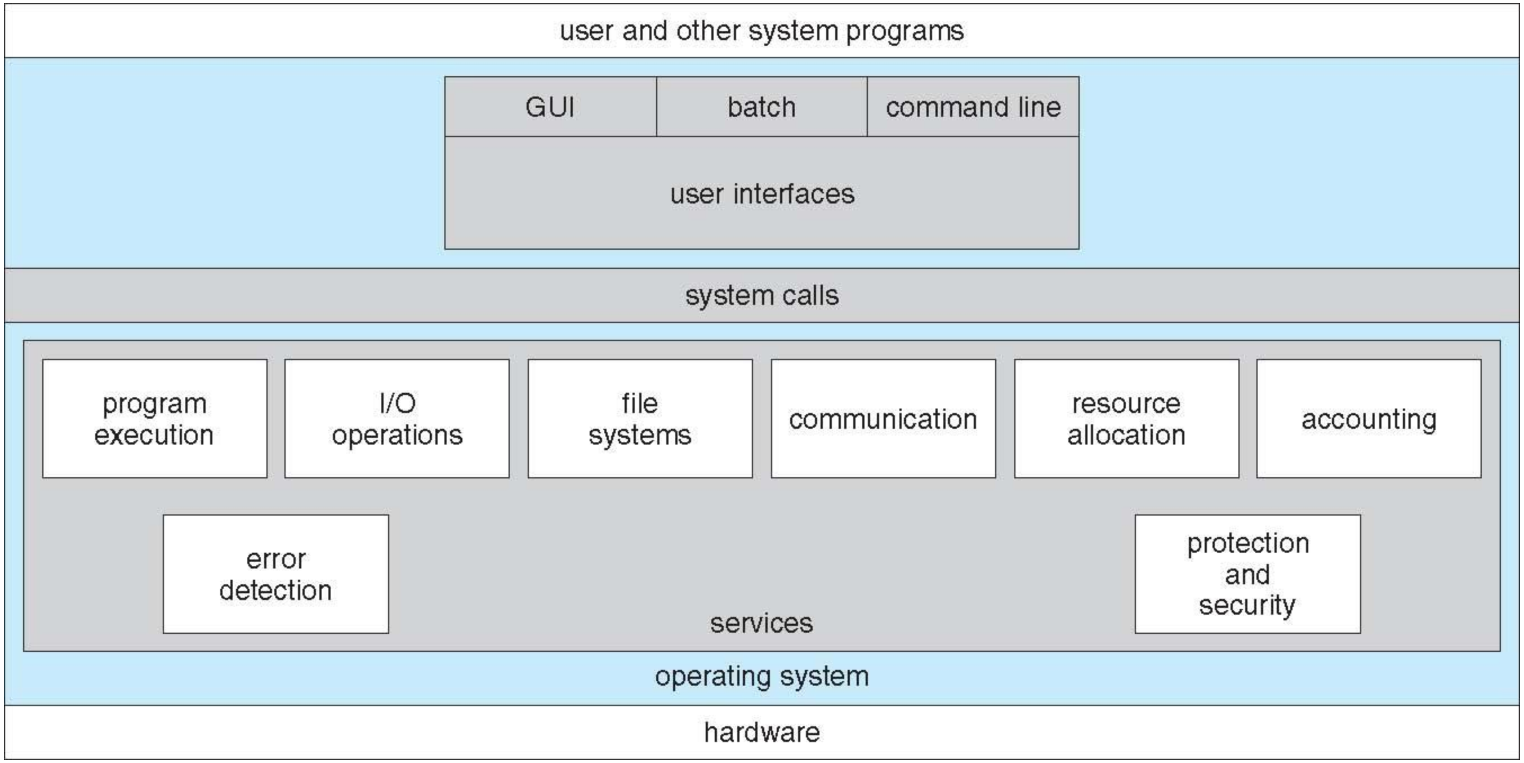


Operating System Services (Cont.)

- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources -   CPU cycles, main memory, file storage, I/O devices.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



A View of Operating System Services

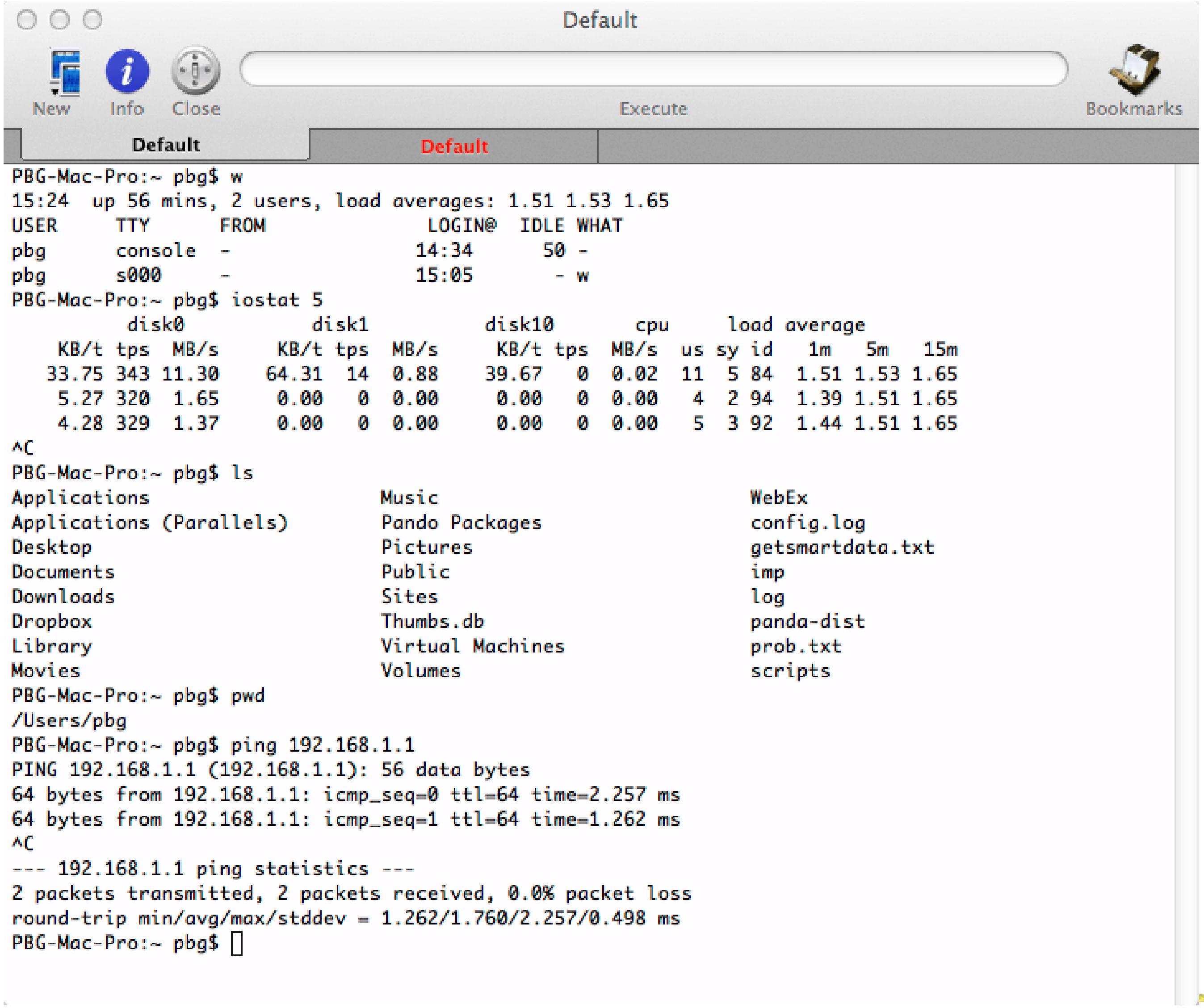


User Operating System Interface - CLI

CLI or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification
  - Unix example
    - rm file.txt
  - In this way, programmers can add new commands to the system easily by creating new files with the proper names

Bourne Shell Command Interpreter

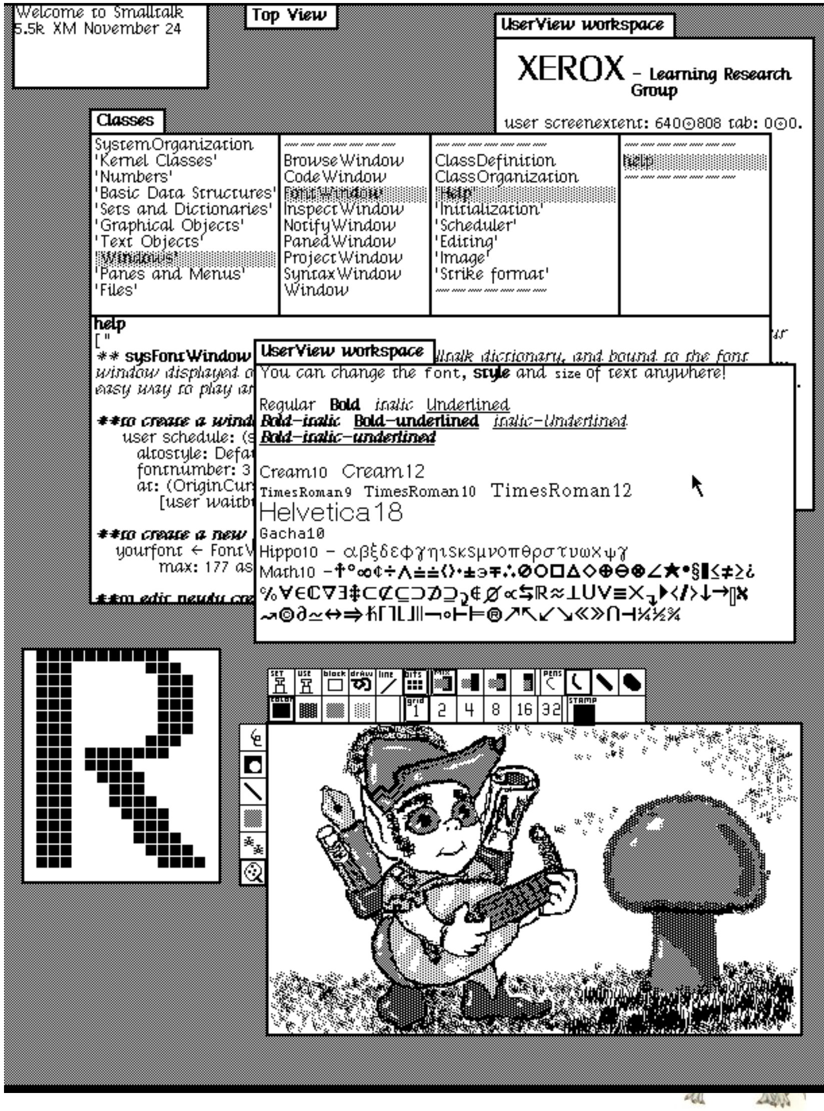


```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10       cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ ▯
```

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 with CLI “command” shell
  - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

XEROX PARC



Welcome to Smalltalk
5.5k XM November 24

XEROX – Learning Research Group

user screenextent: 640⊙808 tab: 0⊙0.

```
SystemOrganization       BrowseWindow    ClassDefinition       help
'Kernel Classes'         CodeWindow      ClassOrganization
'Numbers'                FontWindow      'Help'
'Basic Data Structures'  InspectWindow   'Initialization'
'Sets and Dictionaries'  NotifyWindow    'Scheduler'
'Graphical Objects'      PanedWindow     'Editing'
'Text Objects'           ProjectWindow   'Image'
'Windows'                SyntaxWindow    'Strike format'
'Panes and Menus'        Window
'Files'
```

```
help
["
** sysFontWindow          lltalk dictionary, and bound to the font
window displayed a
easy way to play ar

**to create a wind
   user schedule: (s
      altostyle: Defa
      fontnumber: 3
      at: (OriginCur
         [user waitb

**to create a new
   yourfont ← FontV
         max: 177 as

**to edit newly cre
```

You can change the font, **style** and size of text anywhere!

Regular **Bold** *italic* Underlined
Bold-italic **Bold-underlined** *italic-Underlined*
Bold-italic-underlined

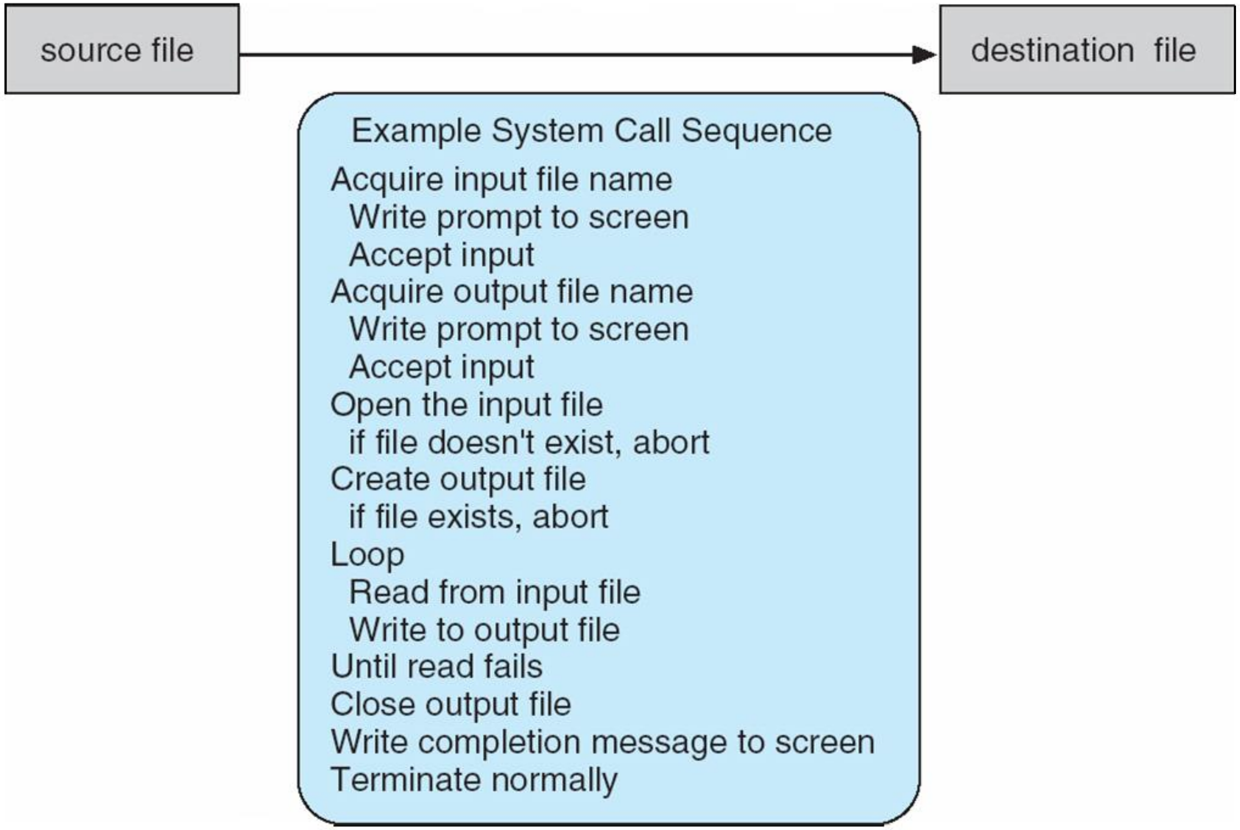
Cream10 Cream12
TimesRoman9 TimesRoman10 TimesRoman12
Helvetica18
Gacha10
Hippo10 – αβξδεφγηιςκςμνοπθρστυωχψγ
Math10 –†°∞¢÷∧≐≐⟨⟩·±∍∓∴ØO□△◇⊕⊖⊗∠★•§▌≤≠≥¿
%∀∈ℂ∇∃⇟⊂⊄⊆⊃⊅⊇⁊∉∅∝⇆ℝ≈⊥∪∨≡×↳▸‹/›↓→▯✘
↝©∂≃↔⇒ℏ⌈⌉⌊⌋‖¬∘⊢⊨®↗↖↙↘«»∩⊣¼½¾

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 Programming 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)

Example of System Calls

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



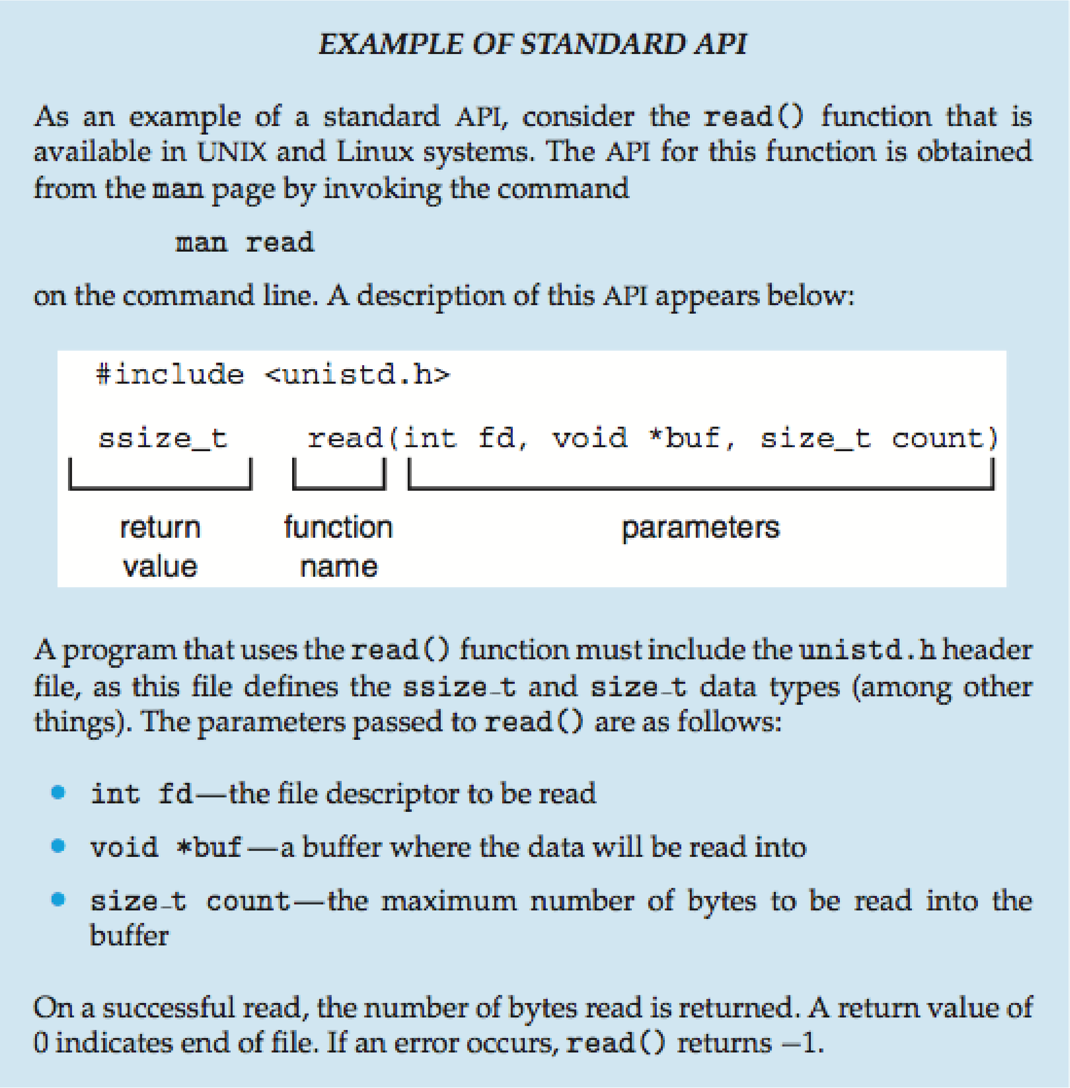
source file

destination file

Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Example of Standard API



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

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

return value

function name

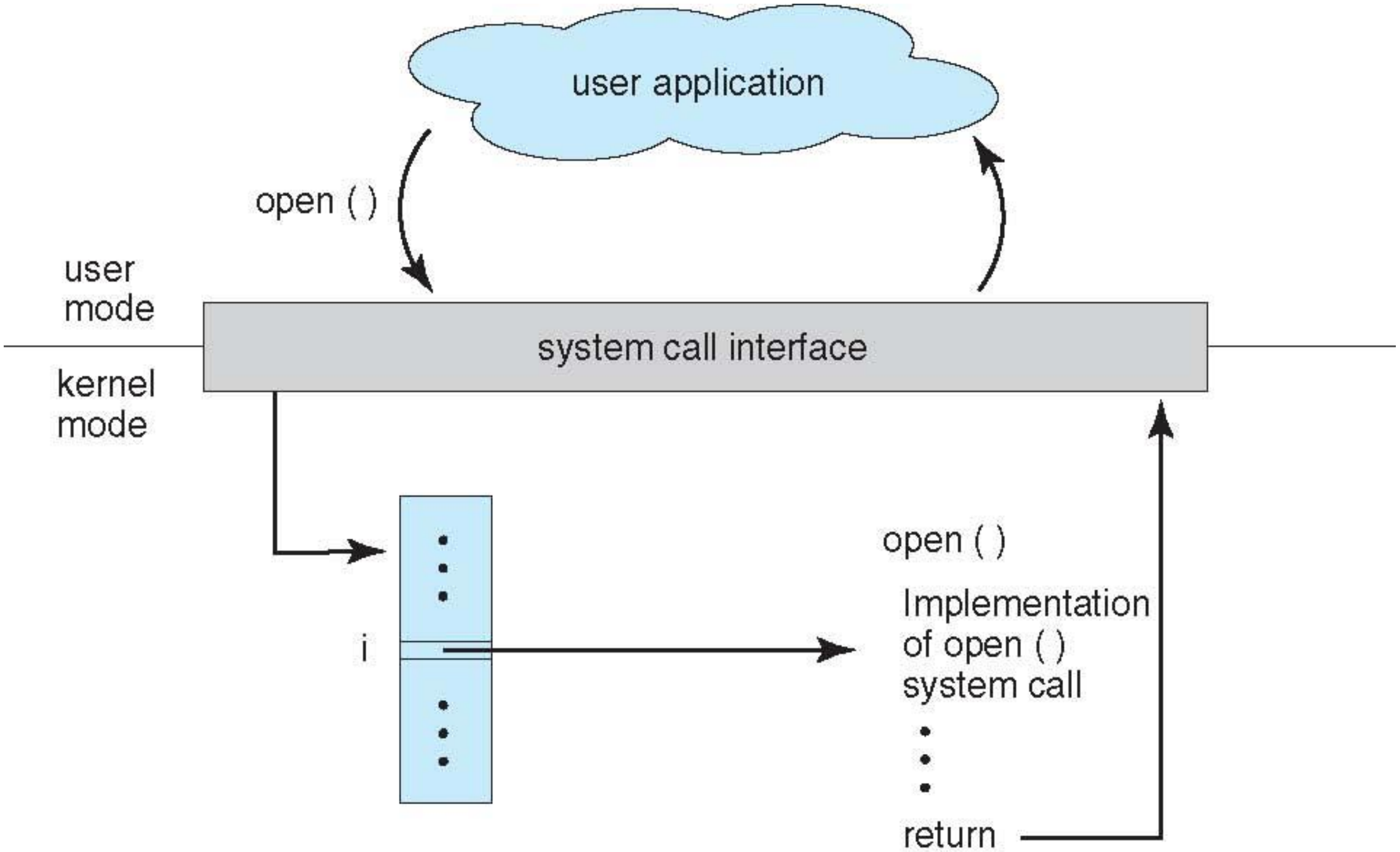
parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

API – System Call – OS Relationship

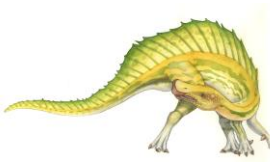


System Call Parameter Passing

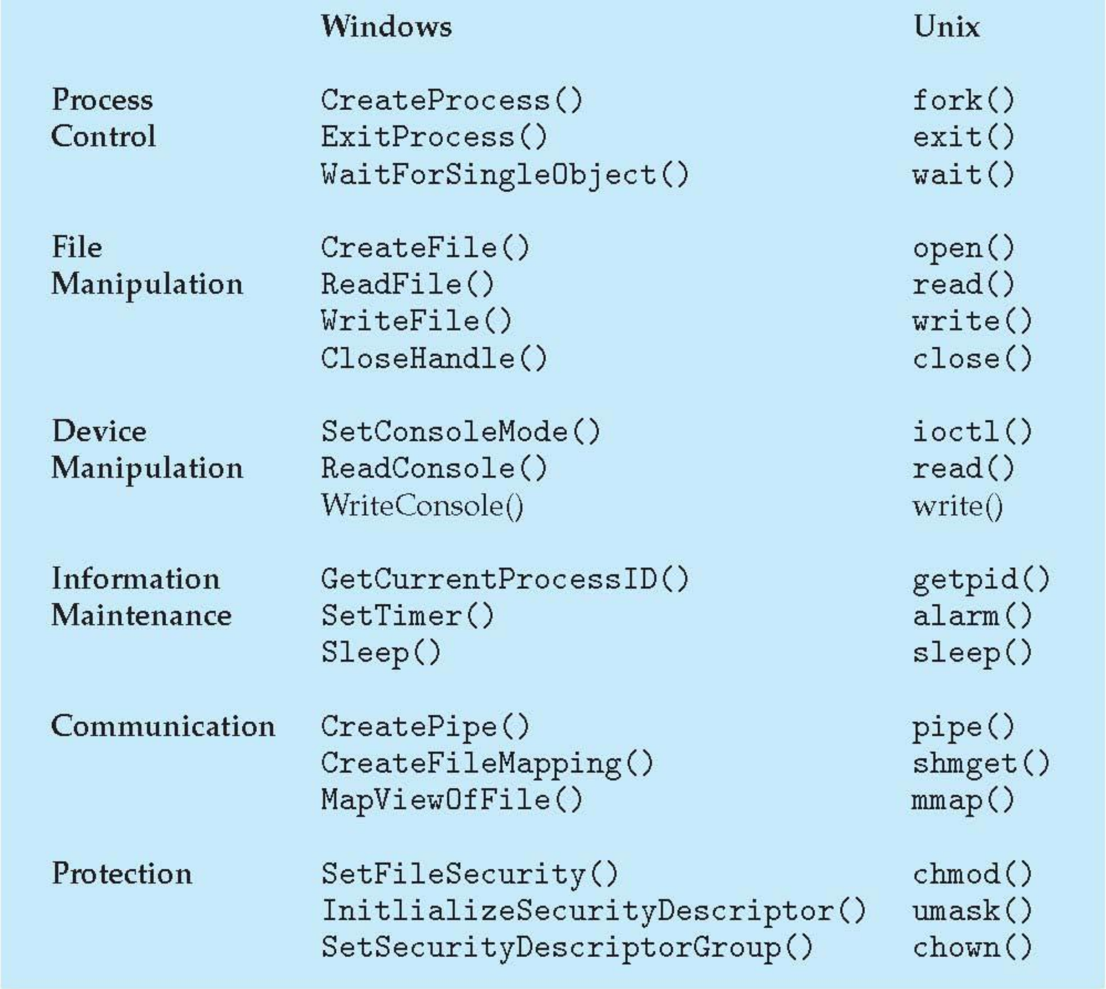
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - **Simplest: pass the parameters in registers**
    - In some cases, may be more parameters 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

Assembly

```
msg  db 'Hello again, World!$'   ; $-terminated message
org  0x100        ; .com files always start 256 bytes into the segment
; int 21h is going to want...

mov  dx, msg      ; the address of or message in dx
mov  ah, 9        ; ah=9 - "print string" sub-function
int  0x21         ; call dos services

mov  dl, 0x0d     ; put CR into dl
mov  ah, 2        ; ah=2 - "print character" sub-function
int  0x21         ; call dos services

mov  dl, 0x0a     ; put LF into dl
mov  ah, 2        ; ah=2 - "print character" sub-function
int  0x21         ; call dos services

mov  ah, 0x4c     ; "terminate program" sub-function
int  0x21         ; call dos services
```



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()

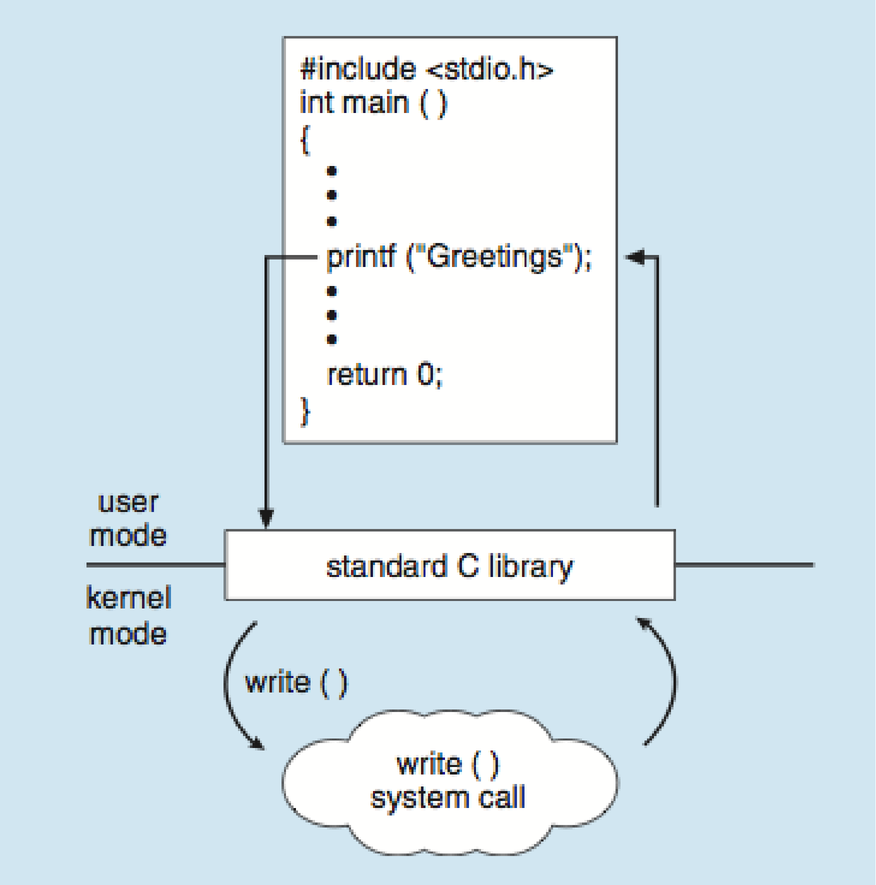
Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

Standard C Library Example

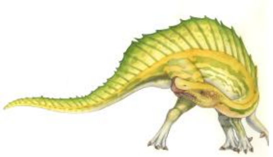
- C program invoking printf() library call, which calls write() system call



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

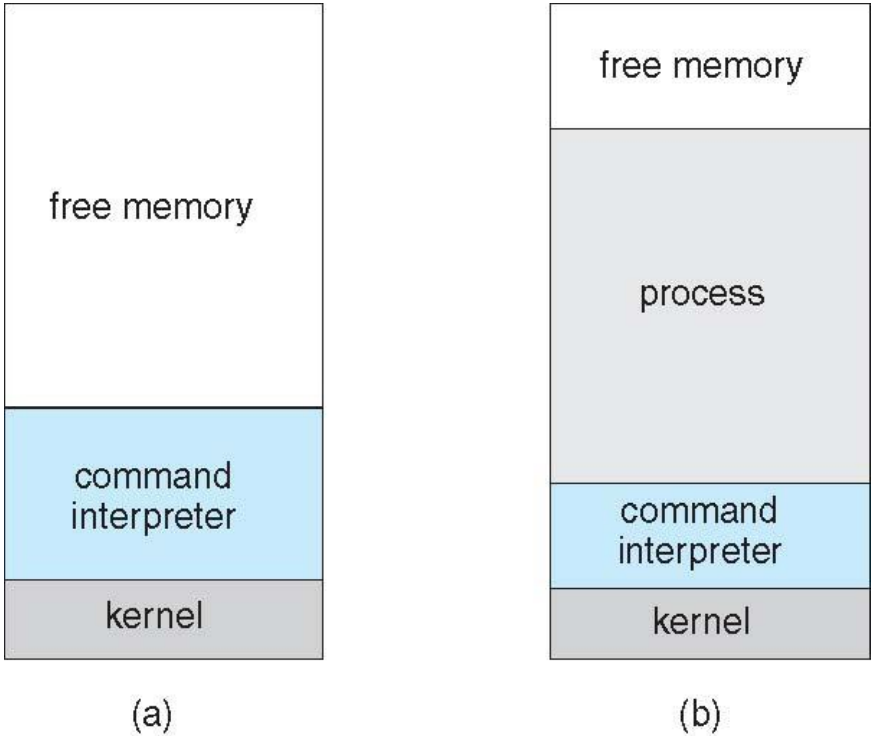
OS Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel –Mach
  - Modules Approach



Example: MS-DOS

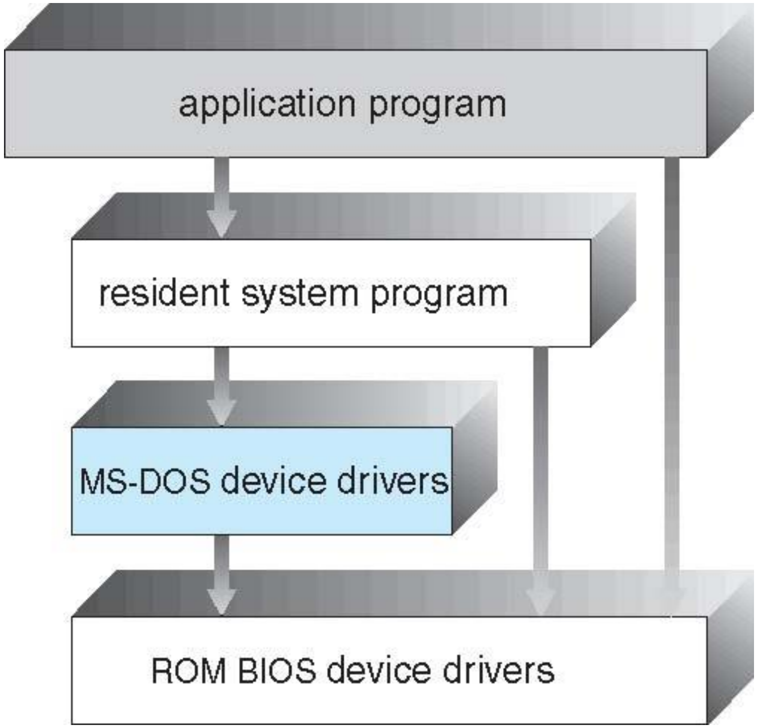
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
    - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup running a program

Simple Structure -- MS-DOS

- 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



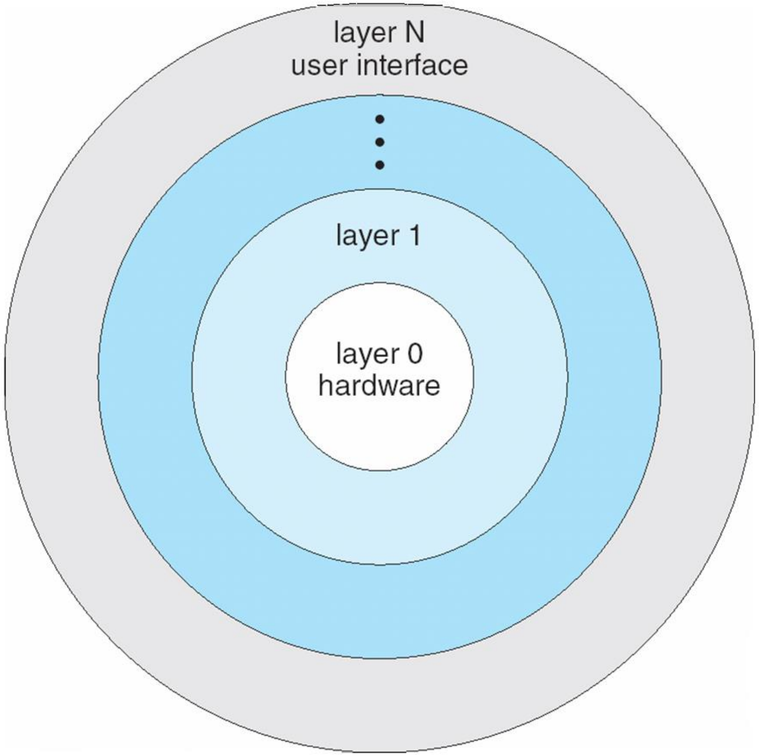
Non Simple Structure -- UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- Systems programs
- 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

Layered Approach

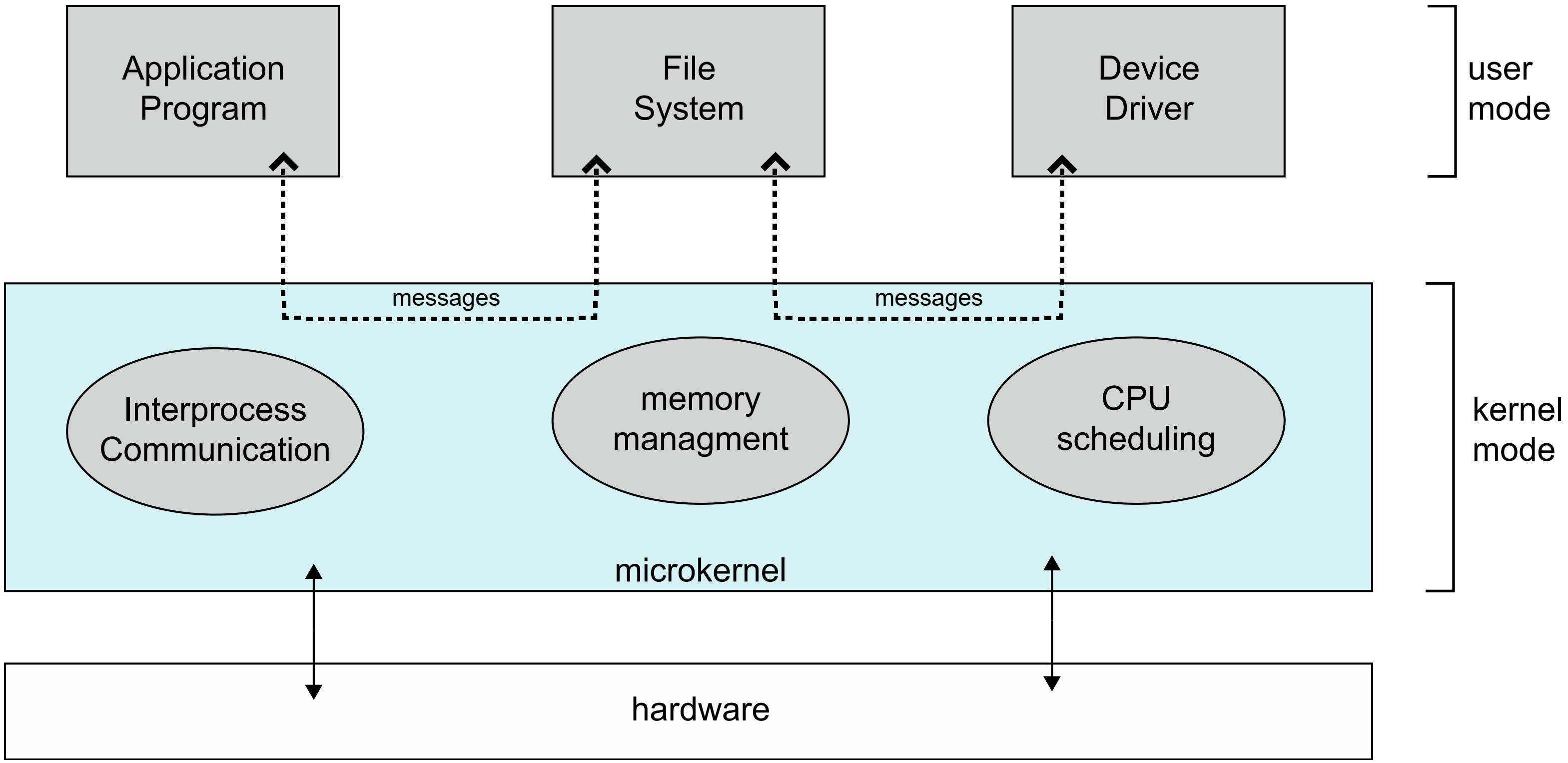
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- 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

Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules. In computing, a loadable kernel module (LKM) is an object file that contains code to extend the running kernel, or so-called base kernel, of an operating system.**
  - 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 with more flexible
  - Most current Unix-like systems and Microsoft Windows support loadable kernel modules, although they might use a different name for them, such as
    - kernel loadable module (kld) in FreeBSD,
    - kernel extension (kext) in macOS,
    - kernel extension module in AIX,
    - kernel-mode driver in Windows NT and downloadable kernel module (DKM) in VxWorks.
    - They are also known as kernel loadable modules (or KLM), and simply as kernel modules (KMOD).
- Disadvantage : **fragmentation penalty!**

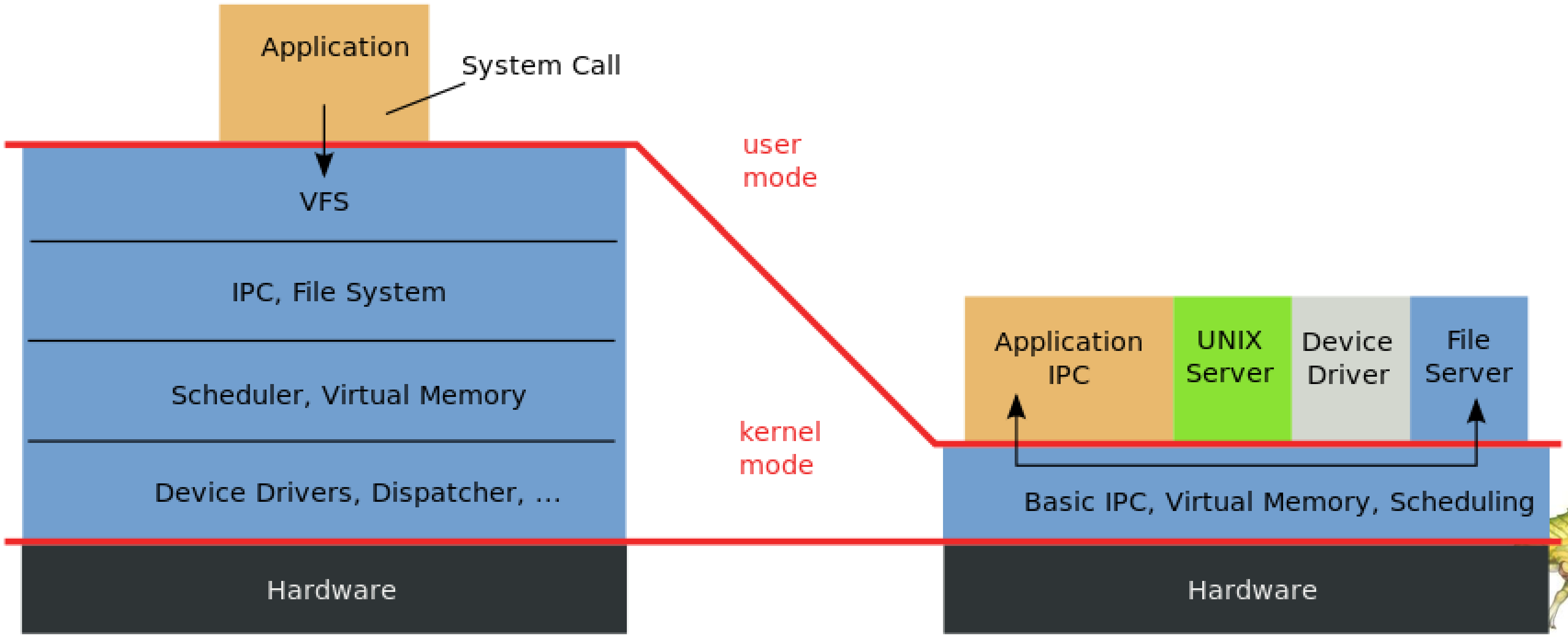
Monolithic vs Microkernel

A **monolithic kernel** is an operating system architecture where the entire operating system is working in kernel space and is alone in supervisor mode

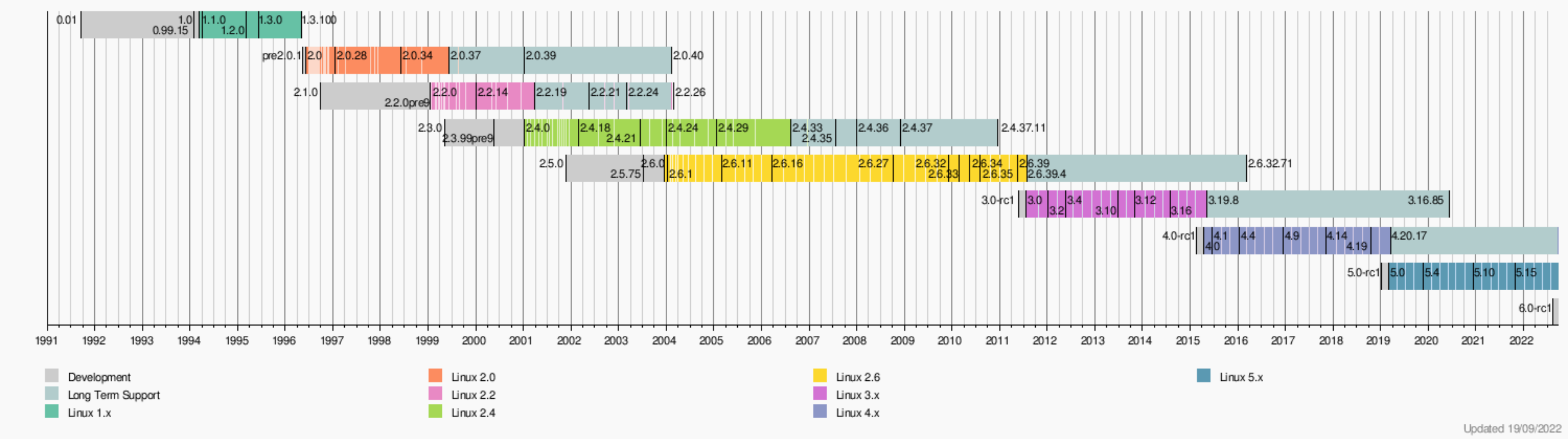
A **microkernel** (also known as **μ-kernel**) is the near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS)

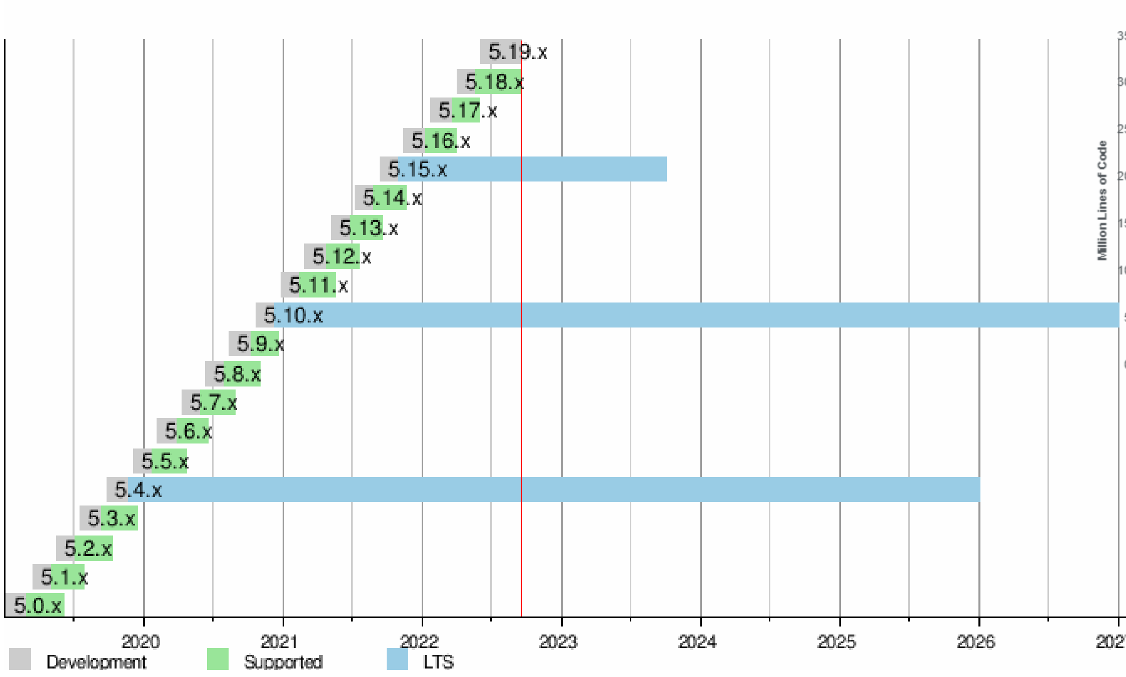
Monolithic Kernel based Operating System

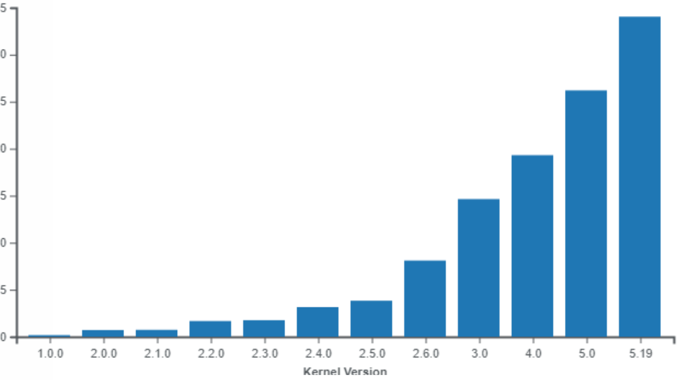
Microkernel based Operating System



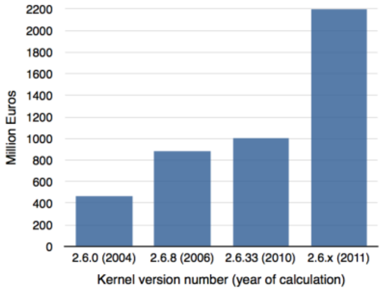
Linux Kernels





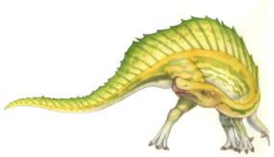


Redevelopment costs of Linux kernel



Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- Failure analysis
  - Log files
    - OS generate **log files** containing error information
  - Core dump
    - **Failure of an application** can generate **core dump** file capturing **memory of the process**
  - Crash dump
    - **Operating system failure** can generate **crash dump** file **containing kernel memory**
- ***Kernighan's Law***
  - "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."



Operating-System Debugging

- Performance tuning
  - Monitor system performance
    - Add code to kernel
    - Use system tools like “top”

End of Chapter 2