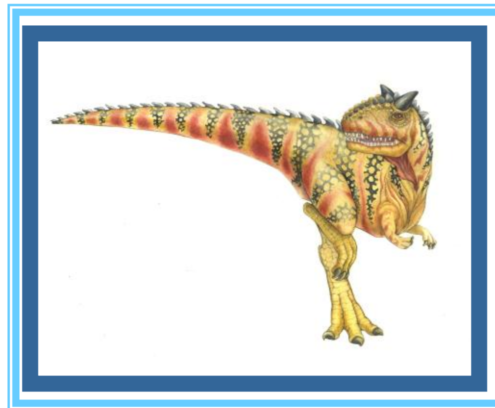
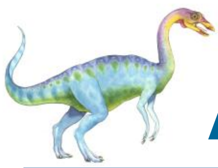
Chapter 2: Operating-System Structures



Silberschatz, Galvin and Gagne ©2009



A View of Operating System Services

user and other system programs

GUI	batch	command line

user interfaces

system calls

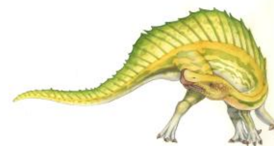
program execution | I/O operations | file systems | communication | resource allocation | accounting

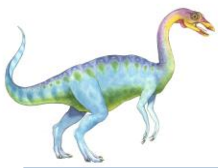
error detection | protection and security

services

operating system

hardware



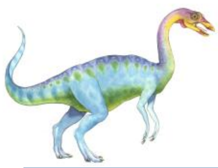


User Operating System Interface

- 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 as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

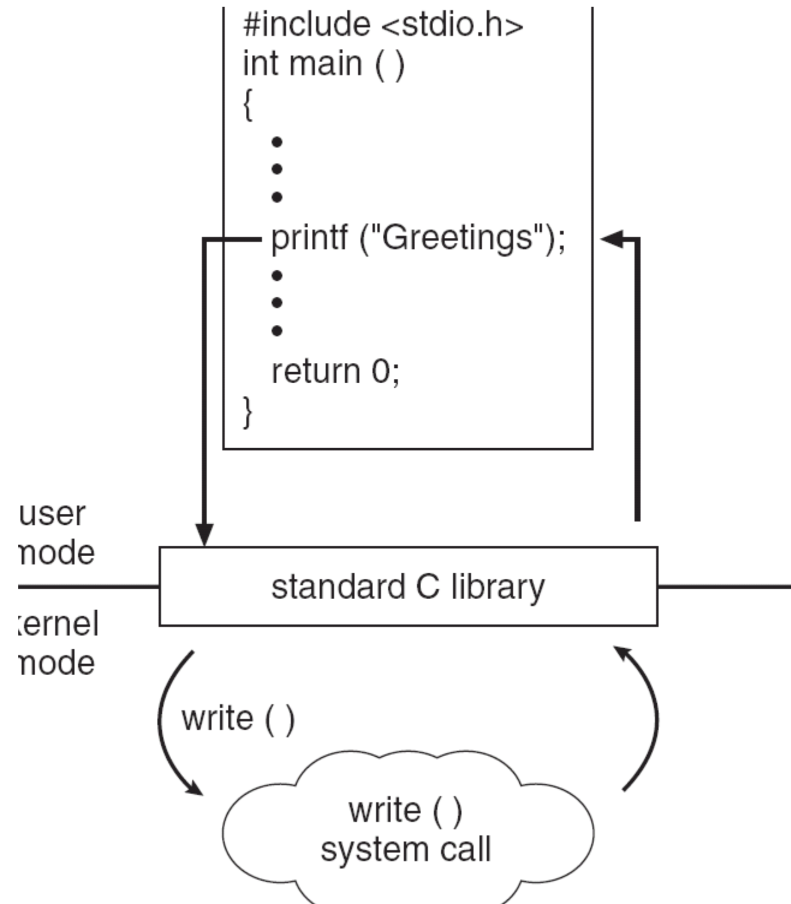
System Calls

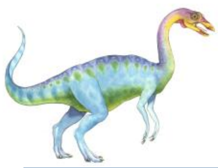
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based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



Standard C Library Example

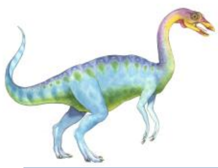
- C program invoking printf() library call, which calls write() system call



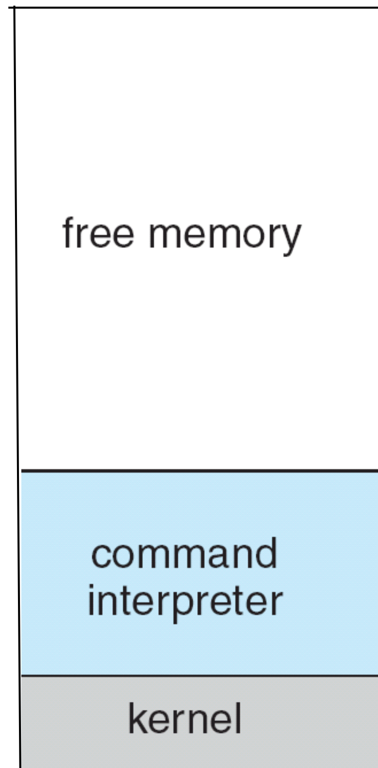


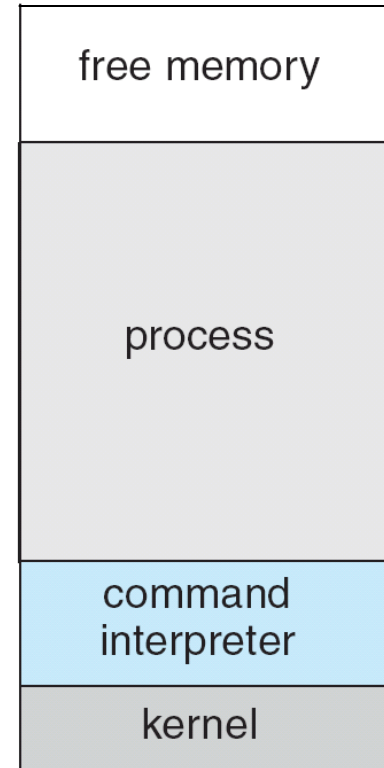
Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

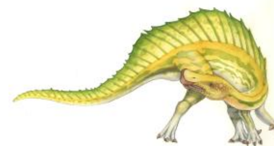


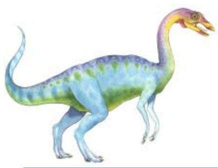
MS-DOS execution



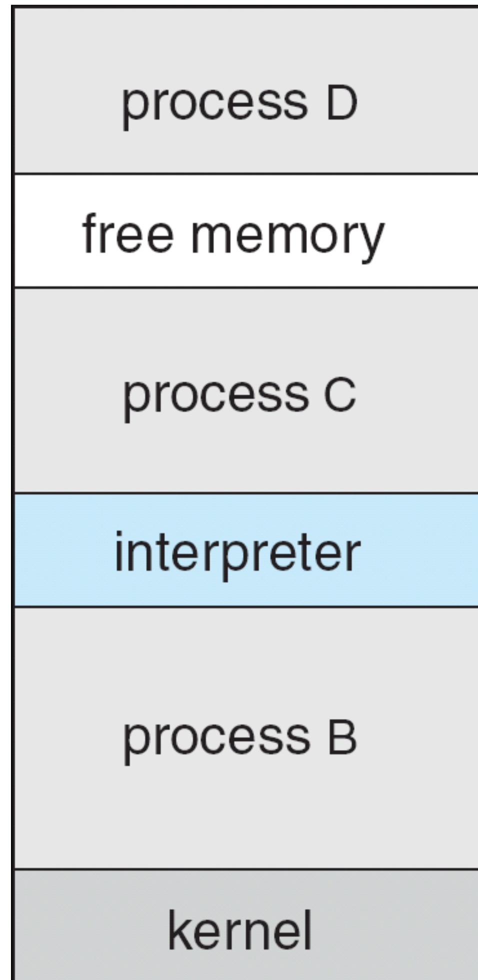


(a) At system startup (b) running a program



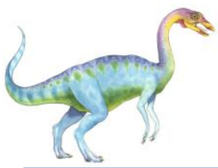


FreeBSD Running Multiple Programs



Simple Structure

- 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



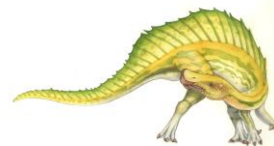
MS-DOS Layer Structure

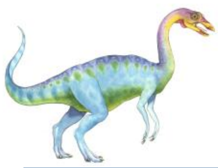
application program

resident system program

MS-DOS device drivers

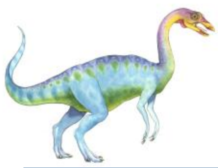
ROM BIOS device drivers





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



Traditional UNIX System Structure

(the users)

shells and commands
compilers and interpreters
system libraries

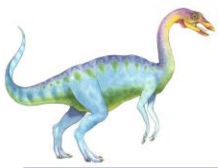
Kernel

system-call interface to the 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

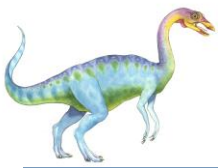
kernel interface to the hardware

terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory



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 with more flexible



Solaris Modular Approach

scheduling classes

device and bus drivers

file systems

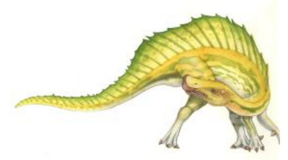
core Solaris kernel

miscellaneous modules

loadable system calls

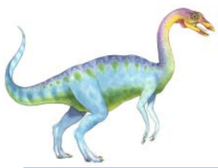
STREAMS modules

executable formats



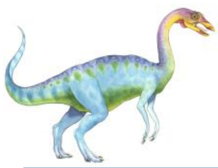
Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. 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



VMware Architecture

application	application	application	application
	guest operating system (free BSD) virtual CPU virtual memory virtual devices	guest operating system (Windows NT) virtual CPU virtual memory virtual devices	guest operating system (Windows XP) virtual CPU virtual memory virtual devices
	virtualization layer		
host operating system (Linux)			
hardware: CPU, memory, I/O devices			



The Java Virtual Machine

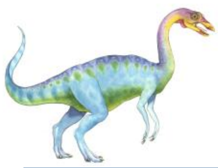
Java program .class files

class loader

Java API .class files

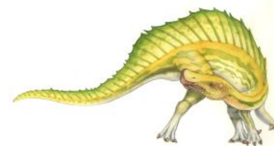
Java interpreter

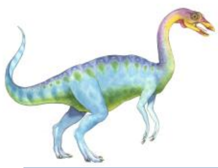
host system (Windows, Linux, etc.)



Dealing with system calls

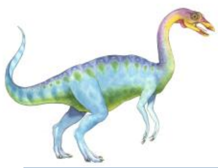
- The programmer should have in and out knowledge of the system call. Like, what exactly it does, system resources it uses, what type of arguments it expects and specially in which cases it fails.
- Most of the linux system calls return an error code if they fail. These error code may vary on the basis of the type of error that caused the failure. So, proper error handling should be in place so that each kind of error should be handled properly and escalated clearly (either to the user or the parent module).
- For the thorough knowledge of system call and the error codes it returns, go through the man page of that specific system call. Man pages are best references to begin with and develop good fundamental understanding about any system call in Linux.





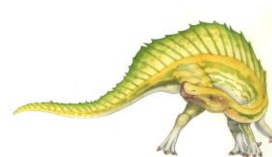
General system call failures

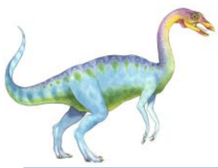
- If a system call tries to access the system hardware and due to any reason the hardware is not available or suppose the hardware is faulty then in that case the system call will fail.
- There are situations when through a system call, a program tries to do a specific task that requires special or root privileges. If the program does not have those kind of privileges then also the system call will fail.
- Passing invalid arguments is another very common reason for system calls to fail.
- Suppose a system call is made to request some memory from heap and due to some reason the system is not able to allocate memory to the requesting process which made the system call, in this case also the system call will fail.



Frequently Used Linux Commands

- Create/extract/view a tar archive.
  - tar cvf archive_name.tar dirname/
  - tar xvf archive_name.tar
  - tar tvf archive_name.tar
- Search string in files
  - grep -i "the" demo_file
  - grep -A 3 -i "example" demo_text
- Find files using file name
  - find -iname "MyCProgram.c“
  - find / -name passwd
  - find -maxdepth 2 -name passwd
  - find -maxdepth 1 -not -iname "MyCProgram.c"





Frequently Used Linux Commands

- Show difference of two files
  - diff -w name_list.txt name_list_new.txt
- Other common commands
  - ls –lh; ls-ltr
  - pwd
  - gzip test.txt; gzip -d test.txt.gz; gzip -l *.gz
  - shutdown -h now
  - ps -ef | more
  - df -h
  - rm -i file*
  - cp -i file1 file2
  - mv -i file1 file2
  - cat file1 file2
  - mkdir ~/temp; mkdir -p dir1/dir2/dir3/dir4/