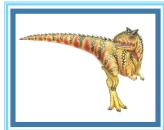


## Chapter 2: System Structures



Operating System Concepts – 9<sup>th</sup> Edition

Silberschatz, Galvin and Gagne ©2013



### Chapter 2: Operating System Structures

- Operating System Services
- System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines



Silberschatz, Galvin and Gagne ©2013

2.2



### Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system

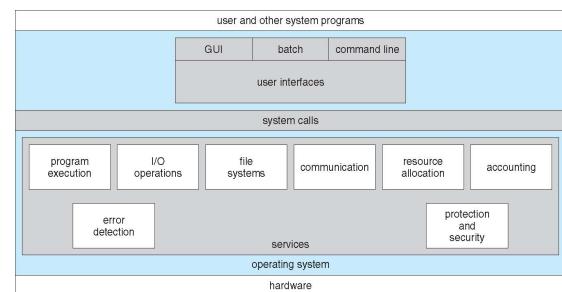
Operating System Concepts – 9<sup>th</sup> Edition

2.3

Silberschatz, Galvin and Gagne ©2013



### A View of Operating System Services



Operating System Concepts – 9<sup>th</sup> Edition

2.4

Silberschatz, Galvin and Gagne ©2013



### Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, permission management.

Operating System Concepts – 9<sup>th</sup> Edition

2.5

Silberschatz, Galvin and Gagne ©2013



### Operating System Services (Cont)

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - **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)
  - **Error detection** - OS needs to be constantly aware of possible errors
    - 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 Concepts – 9<sup>th</sup> Edition

2.6

Silberschatz, Galvin and Gagne ©2013





# 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 common APIs
    - Win32 API for Windows
    - POSIX API for POSIX-based systems (including 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?
    - Hide the complex details of the system call from users
    - Program portability

Operating System Concepts – 9<sup>th</sup> Edition

2.13

Silberschatz, Galvin and Gagne ©2013

## Example of System Calls

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

2

Silberschatz, Galvin and Gagne ©2013

## 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**

```
#include <unistd.h>
ssize_t      read(int fd, void *buf, size_t count)
```

return value	function name	parameters
--------------	---------------	------------

- **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

Operating System Concepts – 9<sup>th</sup> Edition

2.15

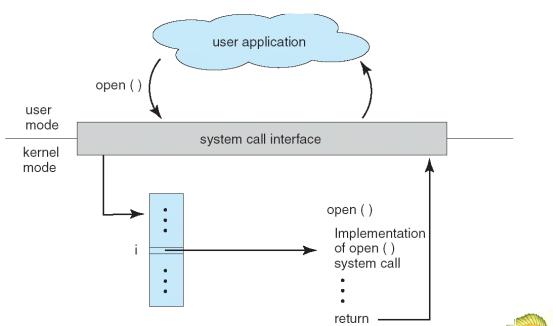
Silberschatz, Galvin and Gagne ©2013

## 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 call
    - Most details of OS interface hidden from programmer by API

Silberschatz, Galvin and Gagne ©2013

API – System Call – OS Relationship



Counting Custom Concepts - 2nd Edition

217

Silberechitz, Galvin and Gagné ©2013

## System Call Parameter Passing

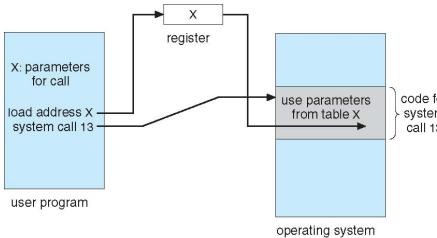
- Often, more information is required than simply the identity of desired system call
    - Exact type and amount of information vary according to OS and call
  - Three general methods used to pass parameters to the OS
    - Simplest: pass the parameters in registers
      - ▶ In some cases, may be more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

Silbersehatz, Galvin and Gagne ©2013

Silbersehatz, Galvin and Gagne ©2013



## Parameter Passing via Table

Operating System Concepts – 9<sup>th</sup> Edition

2.19

Silberschatz, Galvin and Gagne ©2013



## 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

Operating System Concepts – 9<sup>th</sup> Edition

2.20

Silberschatz, Galvin and Gagne ©2013

Operating System Concepts – 9<sup>th</sup> Edition

2.19

Silberschatz, Galvin and Gagne ©2013



## 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

Operating System Concepts – 9<sup>th</sup> Edition

2.21

Silberschatz, Galvin and Gagne ©2013



## Types of System Calls (Cont.)

### Communications

- create, delete communication connection
- send, receive messages if **message passing model** to **host name** or **process name**
  - From **client** to **server**
- **Shared-memory model** create and gain access to memory regions
- transfer status information
- attach and detach remote devices

### Protection

- Control access to resources
- Get and set permissions
- Allow and deny user access

Operating System Concepts – 9<sup>th</sup> Edition

2.22

Silberschatz, Galvin and Gagne ©2013

Operating System Concepts – 9<sup>th</sup> Edition

2.21

Silberschatz, Galvin and Gagne ©2013



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

Silberschatz, Galvin and Gagne ©2013

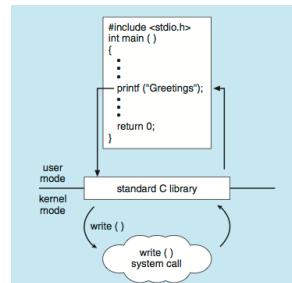
Operating System Concepts – 9<sup>th</sup> Edition

2.23



## Standard C Library Example

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

Operating System Concepts – 9<sup>th</sup> Edition

2.24

Silberschatz, Galvin and Gagne ©2013

Operating System Concepts – 9<sup>th</sup> Edition

2.24

Silberschatz, Galvin and Gagne ©2013



## System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
  
- Most users' view of an operation system is defined by system programs, not the actual system calls

Operating System Concepts – 9<sup>th</sup> Edition

2.25

Silberschatz, Galvin and Gagne ©2013



## System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
  
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
  
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

Operating System Concepts – 9<sup>th</sup> Edition

2.26

Silberschatz, Galvin and Gagne ©2013



## System Programs (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
  
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
  
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
  
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operating System Concepts – 9<sup>th</sup> Edition

2.27

Silberschatz, Galvin and Gagne ©2013



## System Programs (Cont.)

- **Background Services**
  - Launch at boot time
    - ▶ Some for system startup, then terminate
    - ▶ Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services, subsystems, daemons**
  
- **Application programs**
  - Don't pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Operating System Concepts – 9<sup>th</sup> Edition

2.28

Silberschatz, Galvin and Gagne ©2013



## Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- 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, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Concepts – 9<sup>th</sup> Edition

2.29

Silberschatz, Galvin and Gagne ©2013



## Operating System Design and Implementation (Cont.)

- Important principle to separate
  - Policy:** *What* will be done?
  - Mechanism:** *How* to do it?
  
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
  
- Specifying and designing OS is highly creative task of **software engineering**

Operating System Concepts – 9<sup>th</sup> Edition

2.30

Silberschatz, Galvin and Gagne ©2013





## Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower

Operating System Concepts – 9<sup>th</sup> Edition

2.31

Silberschatz, Galvin and Gagne ©2013



## Operating System Structure

- General-purpose OS is very large program
- Various ways to structure one as follows

Operating System Concepts – 9<sup>th</sup> Edition

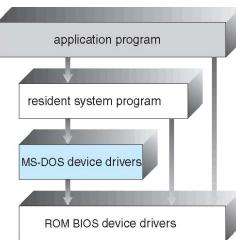
2.32

Silberschatz, Galvin and Gagne ©2013



## Simple Structure

- I.e. 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

Operating System Concepts – 9<sup>th</sup> Edition

2.33

Silberschatz, Galvin and Gagne ©2013



## 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

Operating System Concepts – 9<sup>th</sup> Edition

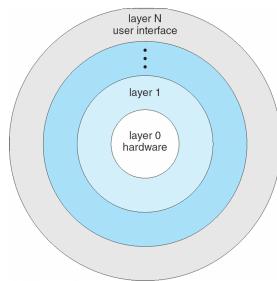
2.34

Silberschatz, Galvin and Gagne ©2013



## 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



Silberschatz, Galvin and Gagne ©2013

Operating System Concepts – 9<sup>th</sup> Edition

2.35



## 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

Operating System Concepts – 9<sup>th</sup> Edition

2.36

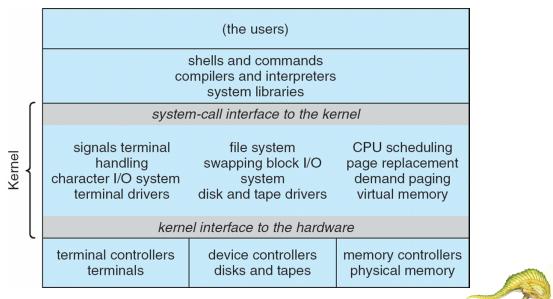
Silberschatz, Galvin and Gagne ©2013





## Traditional UNIX System Structure

Beyond simple but not fully layered



Operating System Concepts – 9<sup>th</sup> Edition

2.37

Silberschatz, Galvin and Gagne ©2013



## 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

Operating System Concepts – 9<sup>th</sup> Edition

2.38

Silberschatz, Galvin and Gagne ©2013



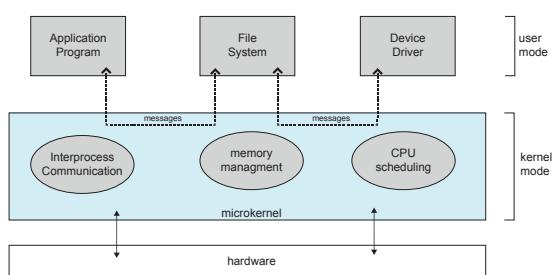
Operating System Concepts – 9<sup>th</sup> Edition

2.37

Silberschatz, Galvin and Gagne ©2013



## Microkernel System Structure



Operating System Concepts – 9<sup>th</sup> Edition

2.39

Silberschatz, Galvin and Gagne ©2013



## Modules

- Most modern operating systems implement **loadable 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
  - Linux, Solaris, etc

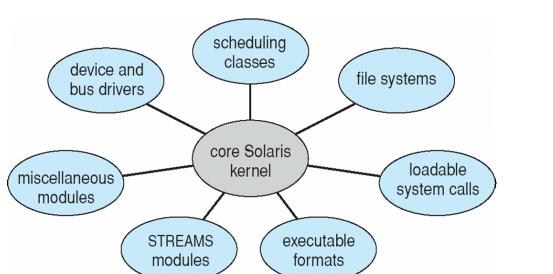
Operating System Concepts – 9<sup>th</sup> Edition

2.40

Silberschatz, Galvin and Gagne ©2013



## Solaris Modular Approach



Operating System Concepts – 9<sup>th</sup> Edition

2.41

Silberschatz, Galvin and Gagne ©2013



## Hybrid Systems

- Most modern operating systems actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem **personalities**
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

Operating System Concepts – 9<sup>th</sup> Edition

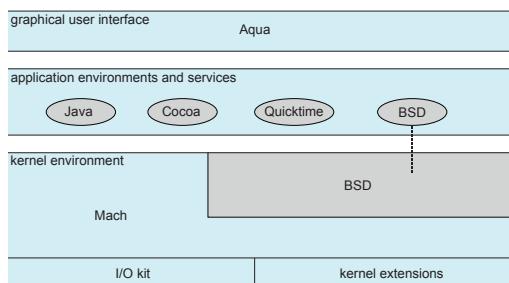
2.42

Silberschatz, Galvin and Gagne ©2013





## Mac OS X Structure

Operating System Concepts – 9<sup>th</sup> Edition

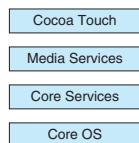
2.43

Silberschatz, Galvin and Gagne ©2013



## iOS

- Apple mobile OS for *iPhone, iPad*
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - ▶ Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Silberschatz, Galvin and Gagne ©2013



## Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to iOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - ▶ Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

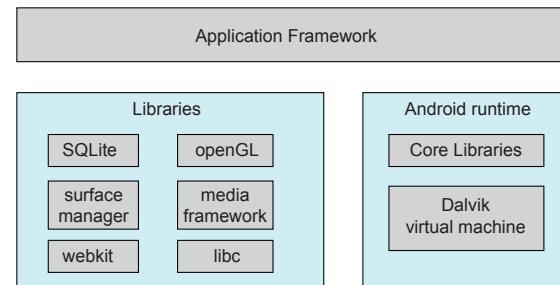
Operating System Concepts – 9<sup>th</sup> Edition

2.45

Silberschatz, Galvin and Gagne ©2013



## Android Architecture

Operating System Concepts – 9<sup>th</sup> Edition

2.46

Silberschatz, Galvin and Gagne ©2013



## Virtual Machines

- The **virtual machine** creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- Software emulation of an abstract machine
  - Make it look like hardware has features you want
  - Programs from one hardware & OS on another one
- Programming simplicity
  - Each process thinks it has all memory/CPU time
  - Each process thinks it owns all devices
  - Different Devices appear to have same interface
- Fault Isolation
  - Processes unable to directly impact other processes
  - Bugs cannot crash whole machine
- Protection and Portability
  - Java interface safe and stable across many platforms

Operating System Concepts – 9<sup>th</sup> Edition

2.47

Silberschatz, Galvin and Gagne ©2013



## Virtual Machines (Cont)

- 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 WinXP guests, each running applications, all on native WinXP **host** OS
  - **VMM** provides virtualization services

Operating System Concepts – 9<sup>th</sup> Edition

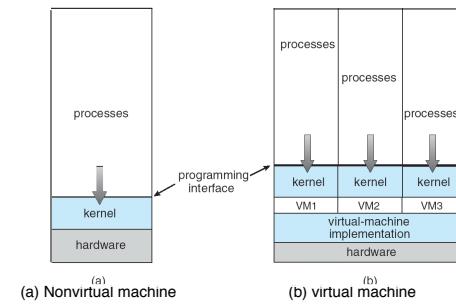
2.48

Silberschatz, Galvin and Gagne ©2013





## Virtual Machines (Cont)

Operating System Concepts – 9<sup>th</sup> Edition

2.49

Silberschatz, Galvin and Gagne ©2013



## Virtual Machines (Cont)

- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  - Apple laptop running Mac OS X host, Windows as a guest
  - Developing apps for multiple OSes without having multiple systems
  - QA testing applications without having multiple systems
  - Executing and managing compute environments within data centers



## Virtual Machines (Cont)

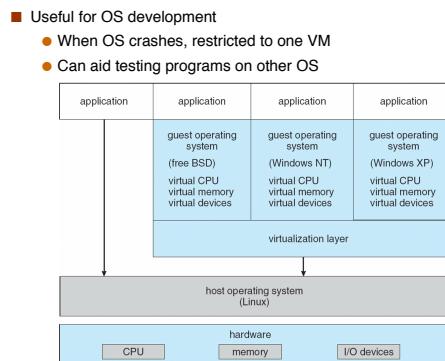
Operating System Concepts – 9<sup>th</sup> Edition

2.50

Silberschatz, Galvin and Gagne ©2013



## Virtual Machines (Cont): Layers of OS

Operating System Concepts – 9<sup>th</sup> Edition

2.51

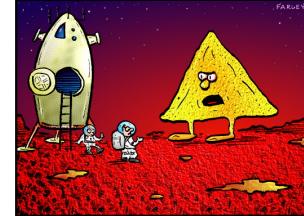
Silberschatz, Galvin and Gagne ©2013



## Nachos: Virtual OS Environment

- You will be working with **Nachos**
  - Simulation environment
  - Hardware, interrupts, I/O
  - Execution of User Programs running on this platform

### DOCTOR FUN



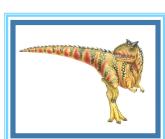
"This is the planet where nachos rule."

2.52

Silberschatz, Galvin and Gagne ©2013



## End of Chapter 2

Operating System Concepts – 9<sup>th</sup> Edition

Silberschatz, Galvin and Gagne ©2013