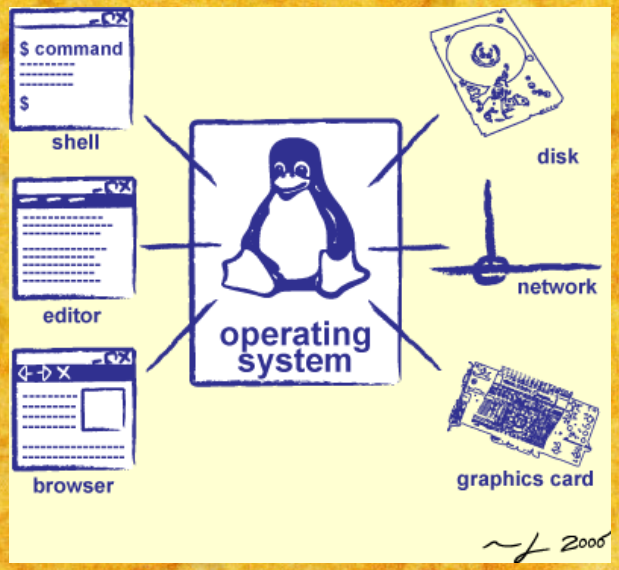
2.- Estructuras de Sistemas Operativos



Estructuras de Sistemas Operativos

- Servicios de SOs
- Interfaz de SOs
- System Calls
- Programas del Sistema
- Diseño e Implementation de SOs
- Estructura de SOs
- Máquinas Virtuales
- Generación de SOs
- Inicio del Sistema (Boot)

Objetivos

- Describir los servicios que el SO proporciona a los usuarios, procesos y otros sistemas
- Estudiar las maneras de estrcturar un SO
- Explicar como se instala un SO, como se personaliza y el proceso de boot

Servicios de un SO

- Servicios útiles al usuario:
  - Interfaz – Casi todo SO tiene una interfaz (UI)
    - Varía entre línea de comandos (CLI), batch, o gráfica (Graphical User Interface – GUI)
  - Ejecución de programas – El SO debe ser capaz de cargar un programa a memoria, ejecutarlo y terminar su ejecución (normal o anormalmente, indicando un error)
  - Operaciones I/O - Un proceso requiere I/O, ya sea de archivos o dispositivos I/O
  - Manipulación de archivos – Los programas necesitan leer y escribir archvos y directorios, crearlos, borrarles, buscar, listar información, manejar permisos, etc.

Servicios de un SO (Cont.)

- Servicios útiles al usuario (Cont.):
  - Comunicaciones – Los procesos intercambian información, en la misma computadora o en la red
    - Vía memoria compartida o por medio de mensajes (paquetes manejados por el SO)
  - Detección de errores
    - En el CPU, memoria, dispositivos de I/O, progrmas, etc.
    - Tomar acciones apropiadas para asegurar cómputo correcto y consistente
    - Las utilerías de depuración (debugging) pueden incrementar la capacidad de los programadores para usar eficientemente el sistema

Servicios de un SO (Cont.)

- Otro conjunto de funciones:
  - **Asignación de recursos** - Múltiples usuarios/jobs concurrentes
    - Tipos de recursos - Algunos (CPU, memoria, archivos) requieren código especial. Otros (dispositivos I/O) puedenn tener código general de solicitud y desalojo
  - **Contabilidad** – Quién usa qué y cuánto.
  - **Protección y seguridad** – Control de quien usa la información. Procesos concurrentes no deben intereferir
    - **Protección** control de acceso a los recursos del sistema
    - **Seguridad** requiere autenticación
    - Si se requiere protección y seguridad, se debe regular. Una cadena es tan fuerte como el más débil de sus eslabones

Interfaz - CLI

CLI – teclear comandos

- Implementado en el kernel o en programas del sistema
- Diferentes formas – **shells**
- Leer comando – ejecutar comando
  - Algunos comandos son “built-in”, otros son nombres de programas
  - Adiciones no requieren modificar el shell

Interfaz - GUI

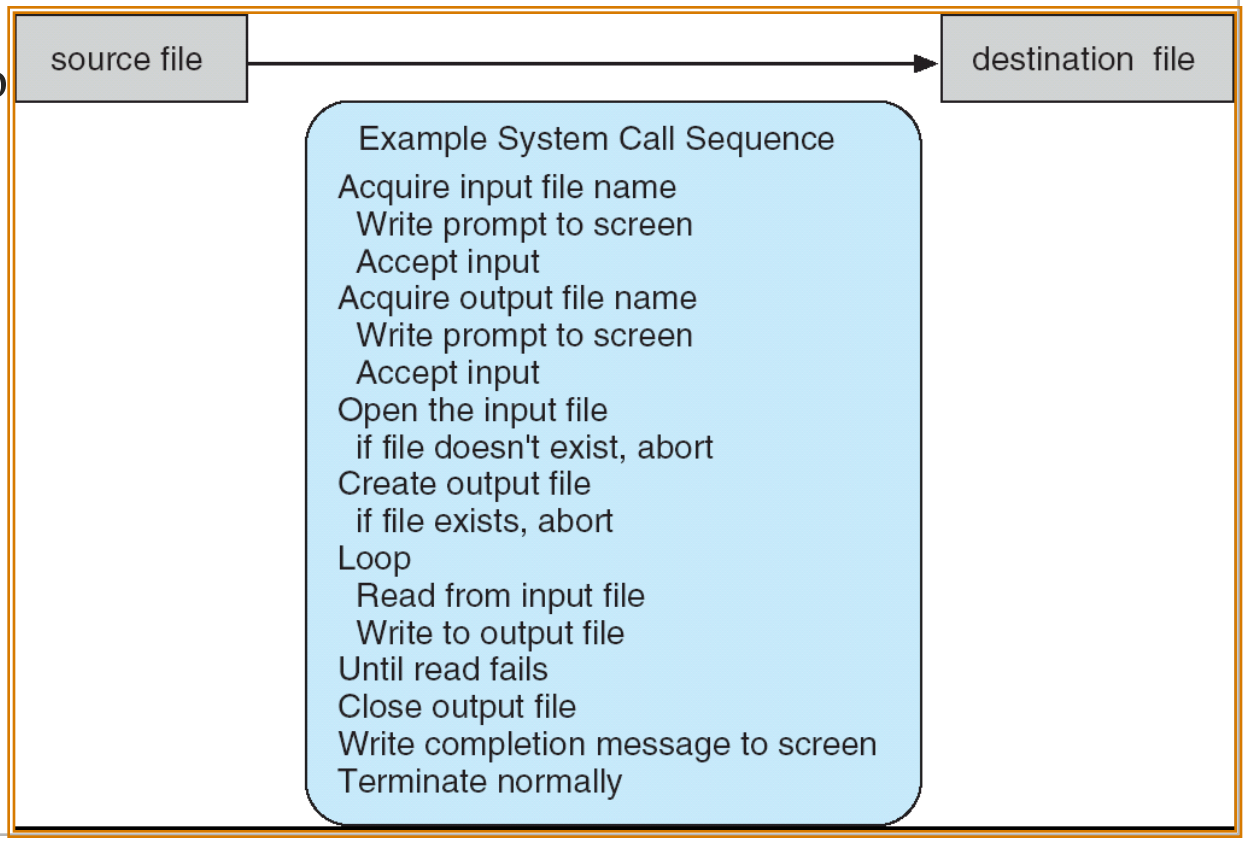
- Interfaz con la metáfora **desktop**
  - Mouse, teclado y monitor
  - **Iconos** representan archivos, programas, acciones, etc
  - Clicks/drags del ratón sobre objetos causan acciones (información, opciones, ejecución, abrir directorio – aka folder, etc.)
  - Inventado en Xerox PARC
- Muchos sistemas incluyen ambas interfaces: CLI y GUI
  - Microsoft Windows es GUI con "command shell" CLI
  - Mac OS X tiene interfaz GUI con kernel UNIX
  - Solaris es CLI con interfaces GUI opcionales (Java Desktop, CDE, KDE)

System Calls

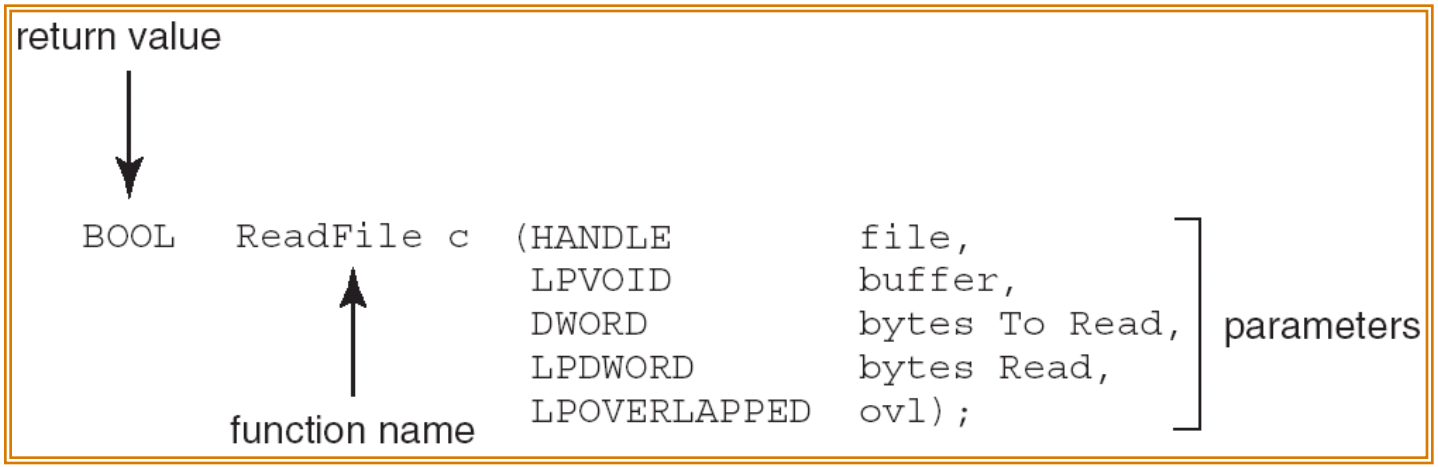
- Interfaz de programación a los servicios proporcionados por el SO
- Típicamente escrita en un lenguaje de alto nivel (C o C++)
- Accesadas por programas vía un **Application Program Interface (API)** de alto nivel, y no directamente
- Las tres APIs más comunes son:
  - Win32 API (Windows)
  - POSIX API (POSIX-based: UNIX, Linux, y Mac OS X)
  - Java API (Java virtual machine JVM)
- ¿Porqué usar APIs en lugar de system calls?

Ejemplo de System Calls

- Secuencia de system calls para copiar el contenido de un archivo a otro



Ejemplo de API

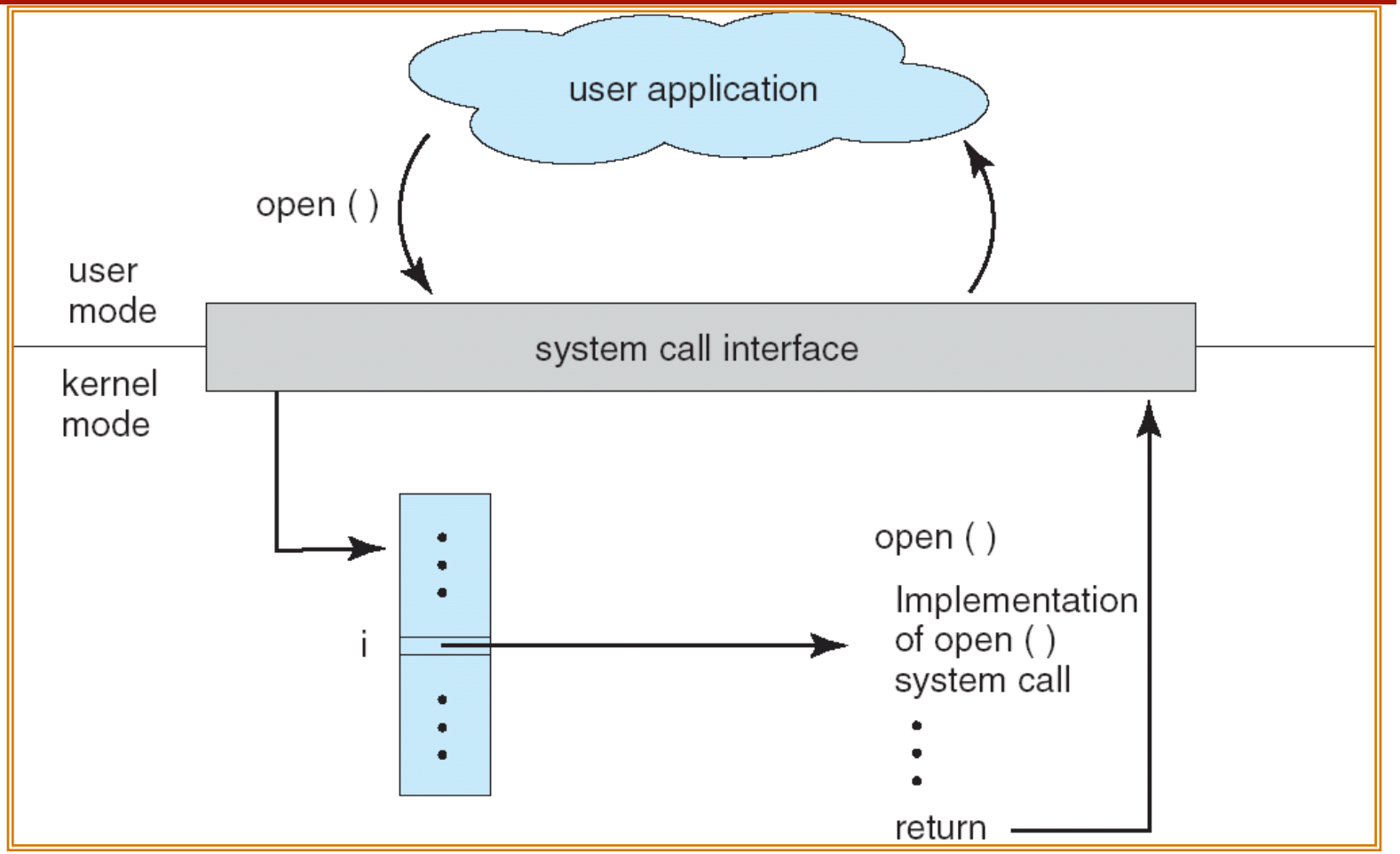


- Descripción de los parámetros pasados a ReadFile()
  - HANDLE file — archivo a leer
  - LPVOID buffer — buffer de donde se leerá y a donde se escribirá
  - DWORD bytesToRead — no. de bytes a leer en el buffer
  - LPDWORD bytesRead — no. de bytes leídos en el último read
  - LPOVERLAPPED ovl — indica si se usa overlapped I/O

Implementación de System Calls

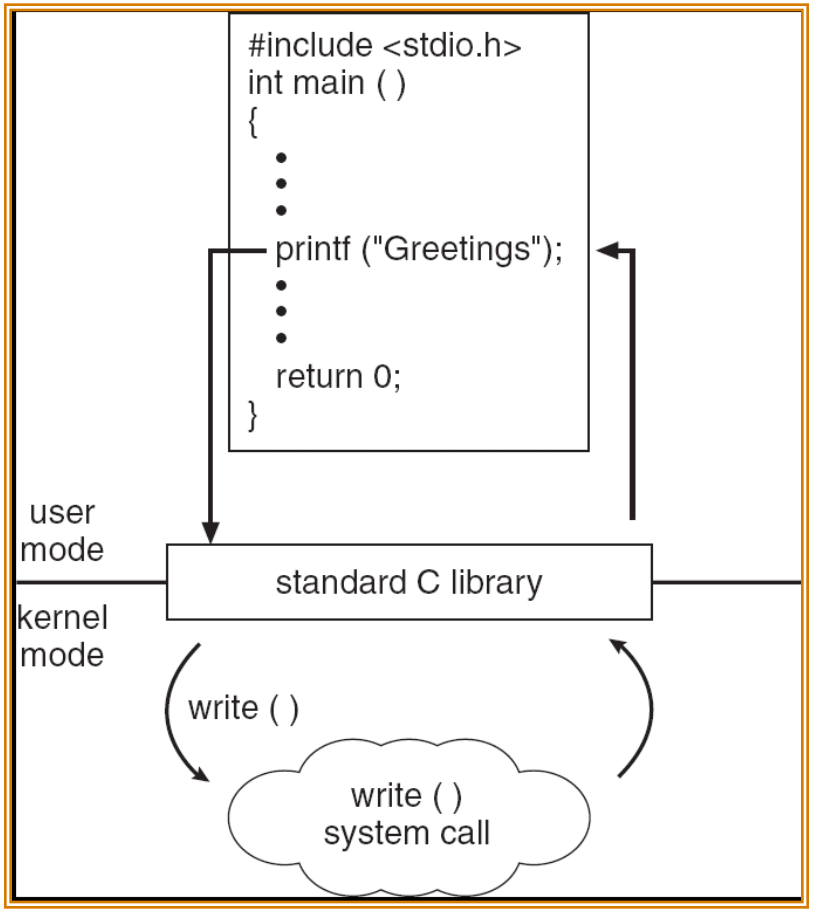
- Típicamente se tiene un no. asociado con cada system call
  - La interfaz system-call mantiene una tabla indexada por no. de system call
- La interfaz system-call invoca los system calls en el kernel del SO y regresa el estatus de terminación y valores de resultados
- No se necesita saber como está implementada la rutina (system call)
  - Seguir el API y entender lo que hace el SO (no como lo hace)
  - Los detalles son ocultados del programador por la API
    - Manejadas por run-time support library (biblioteca de funciones incluidas con el compilador)

Relación API – System Call – OS



Ejemplo de Biblioteca C Estándar

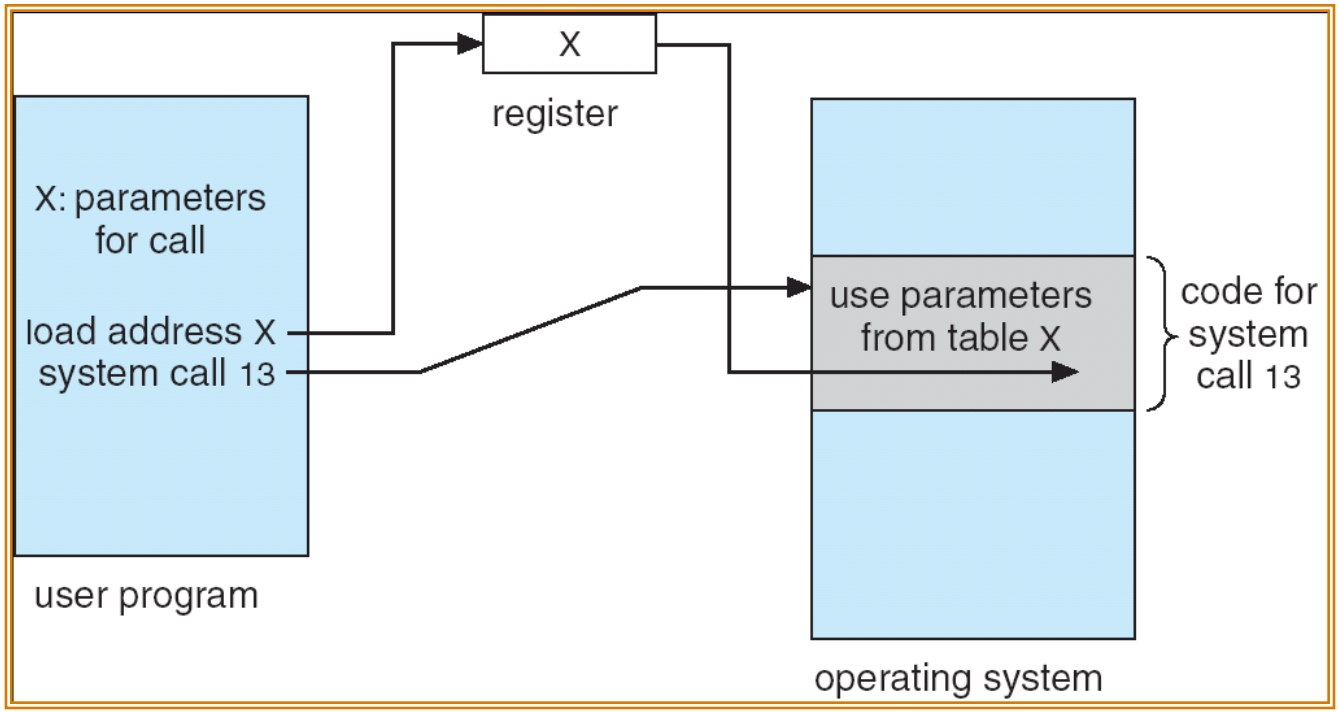
- El programa C invoca printf() (library call), la cual llama a write() (system call)



Paso de Parámetros a un System Call

- Envío de información a un system call
  - Tipo y cantidad de información depende del SO y que SC se está usando
- Métodos de paso de parámetros al SO
  - Pasar parámetros en *registros*
    - Puede haber más parámetros que registros
  - Parámeters almacenados en un *block,* o tabla, en memoria, y la dirección del block se pasa como parámetro en un registro
    - Linux y Solaris
  - Parámetros puestos (*push*), en el *stack* por el programa y sacados (*popped*) por el SO
  - Los métodos block y stack no limitan el número o longitud de los parámetros

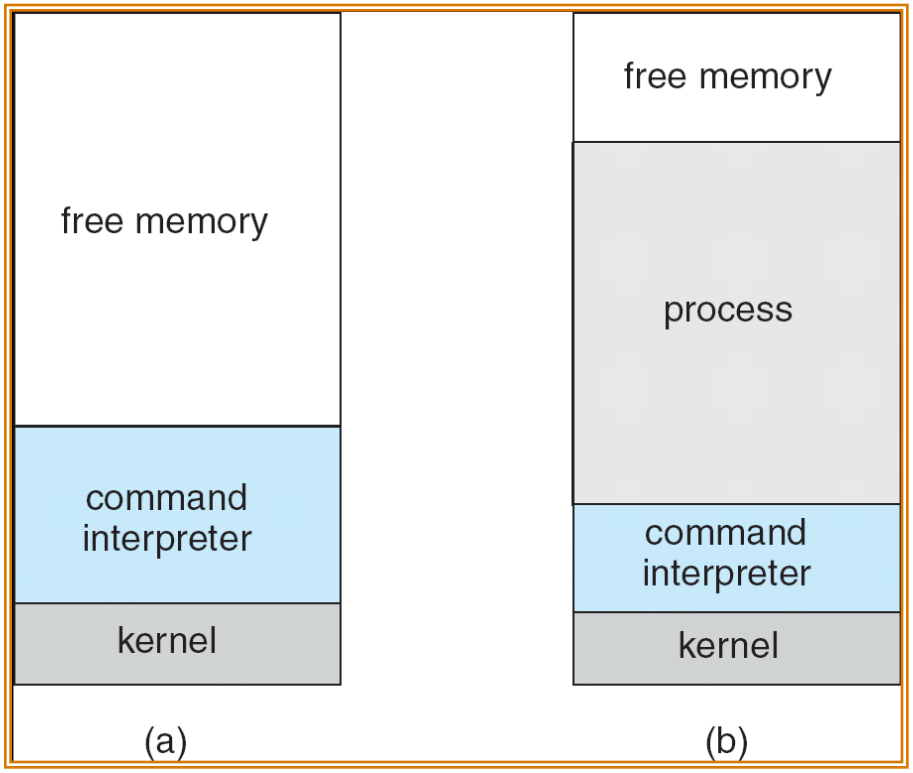
Paso de Parámetros vía Tabla



Tipos de System Calls

- Control de procesos
- Manejo de archivos
- Manejo de dispositivos
- Mantenimiento de información
- Comunicaciones

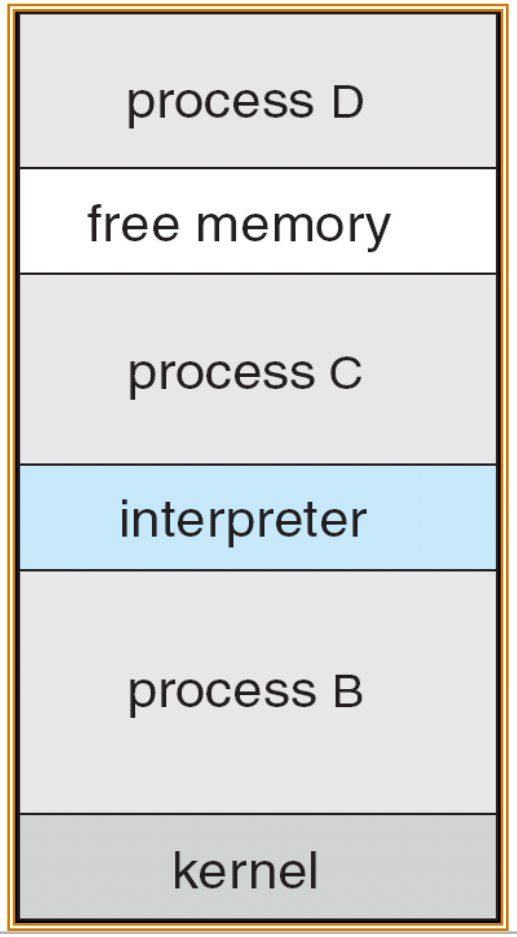
Ejecución en MS-DOS



(a) Al inicio programa

(b) ejecutando un

Programas en Ejecución en FreeBSD



Programas del Sistema

- Los programas del sistema proporcionan un medio ambiente conveniente para desarrollo y ejecución de programas.
- Tipos:
  - Manipulación de archivos
  - Información de status
  - Modificación de archivos
  - Soporte para programación
  - Carga y ejecución de programas
  - Comunicaciones
  - Programas de aplicación
- La mayoría de los usuarios catalogan al SO por los programas del sistema, y no por las system calls

Programas del Sistema

- Proporcionan un medio adecuado para desarrollar y ejecutar programas
  - Algunos solo son interfaces a system calls; otras son mucho mas complejos
- Manejo de archivos - Create, delete, copy, rename, print, dump, list, etc.
- Información del estado
  - Algunos preguntan al SO - date, time, memoria disponible, espacio en disco, no. de usuarios, etc.
  - Información detallada - performance, logging, debugging, etc.
  - Formatean e imprimen a la terminal u otros dispositivos I/O
  - Algunos implementan un registro – usado para almacenar información de configuración

Programas del Sistema (Cont.)

- Modificación de archivos
  - Editores
  - Búsqueda dentro de archivos y transformaciones de texto
- Herramienas para programadores - Compiladores, Ensambladores, debuggers, intérpretes, etc.
- Carga y ejecución de programas – Cargadores absolutos y relocalizables, editores de encadenamiento, cargadores de overlay, debugging para lenguajes de alto nivel y de máquina
- Comunicaciones – mecanismo para crear conexiones virtuales entre procesos, usuarios y computadoras
  - Envío de mensajes entre pantallas, web browsers, email, rlogin, ftp, etc.

Diseño e Implementación de SOs

- Diseño e Implementación de Sos – sin solución, pero hay técnicas de desarrollo exitosas
- La estructura interna de los Sos puede variar mucho
- Definir metas y especificaciones
- Consideraciones: hardware, tipo de sistema
- Metas del *Usuario* - Metas del *Sistema*
  - Metas del usuario – facilidad de uso y aprendizaje, confiable, seguro y rápido
  - Metas del sistema – facilidad de diseño, implementación y mantenimiento, flexible, confiable, sin errores y eficiente

Diseño e Implementación de Sos (Cont.)

- Principio fundamental: Separar

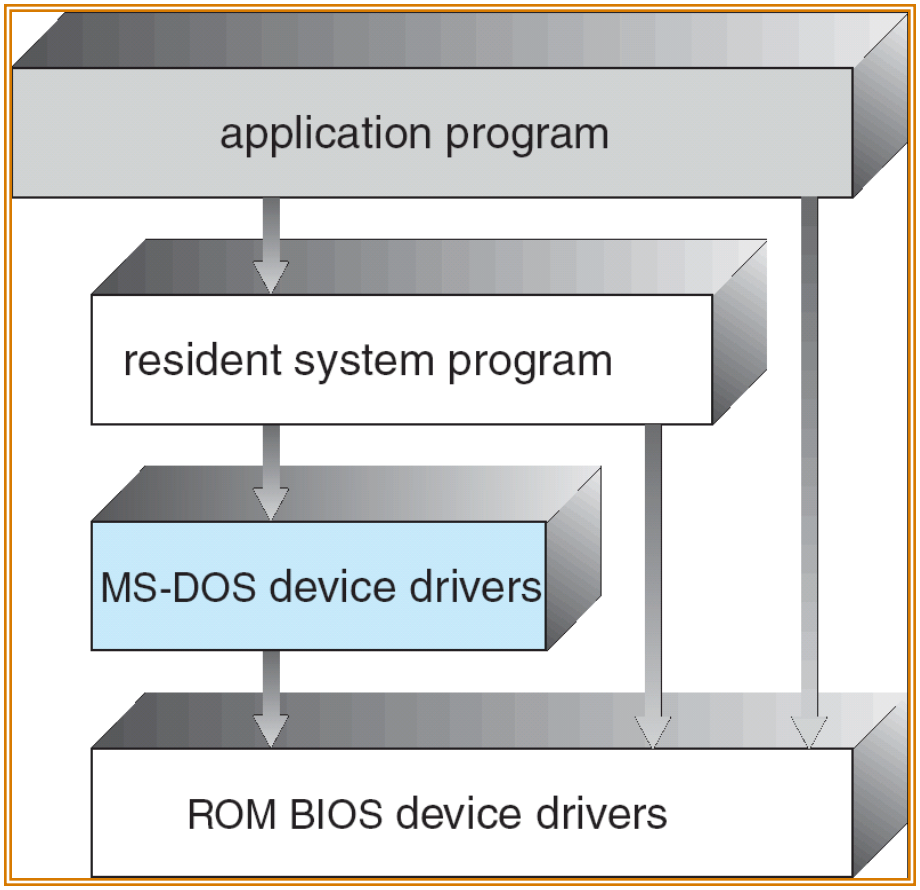
  Política: ¿Qué?
  Mecanismo: ¿Cómo hacerlo?

- Proporciona flexibilidad en el caso de que las políticas cambien

Estructura Simple

- MS-DOS – proporcionar la mayor funcionalidad en el menor espacio
  - Sin división por modules
  - Interfaces y niveles de funcionalidad no están bien separados

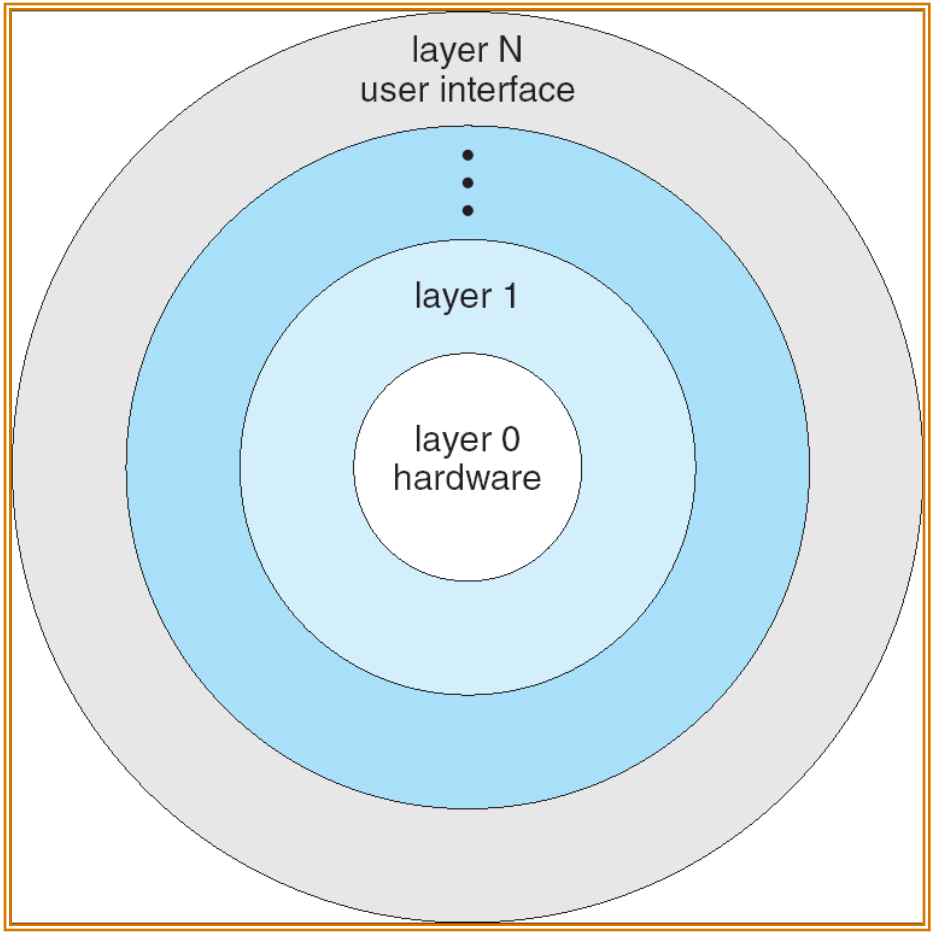
MS-DOS
Estructura de Niveles



Arquitectura Multi-capa

- División por niveles (capas)
- Cada nivel construido sobre otros niveles
- Nivel 0 es hardware
- Nivel N, el mas alto, es UI
- Modularidad. Un nivel solo usa funciones y servicios de niveles más bajos

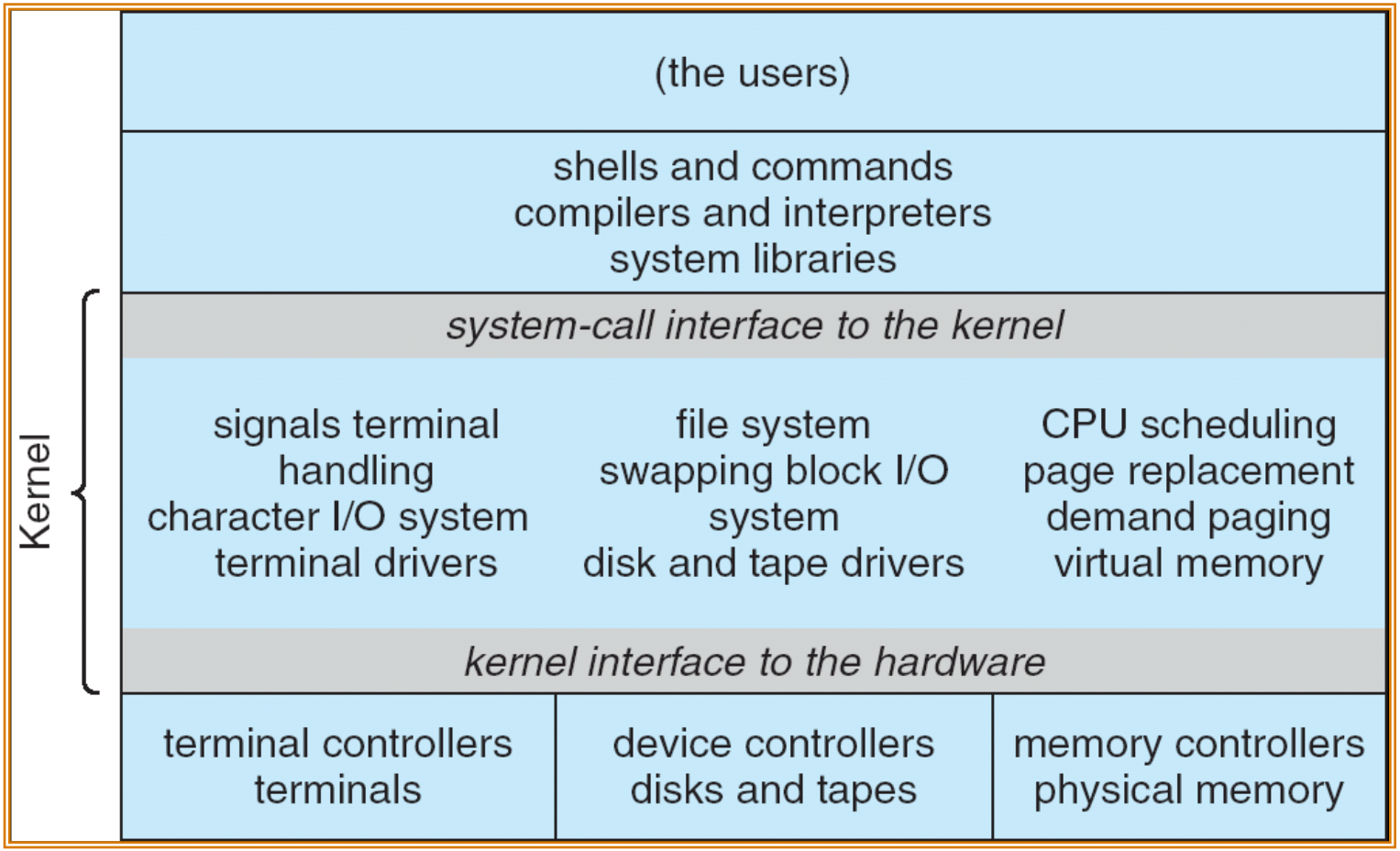
Arquitectura Multi-capa



UNIX

- Limitado por la funcionalidad del hardware.
- Consiste de dos partes
  - Systems programs
  - Kernel
    - Cualquier cosa debajo de la interfaz system-call y arriba del hardware
    - Proporciona el sistema de archivos, CPU scheduling, manejo de memoria y otras funciones.
    - Demasiadas funciones para un nivel

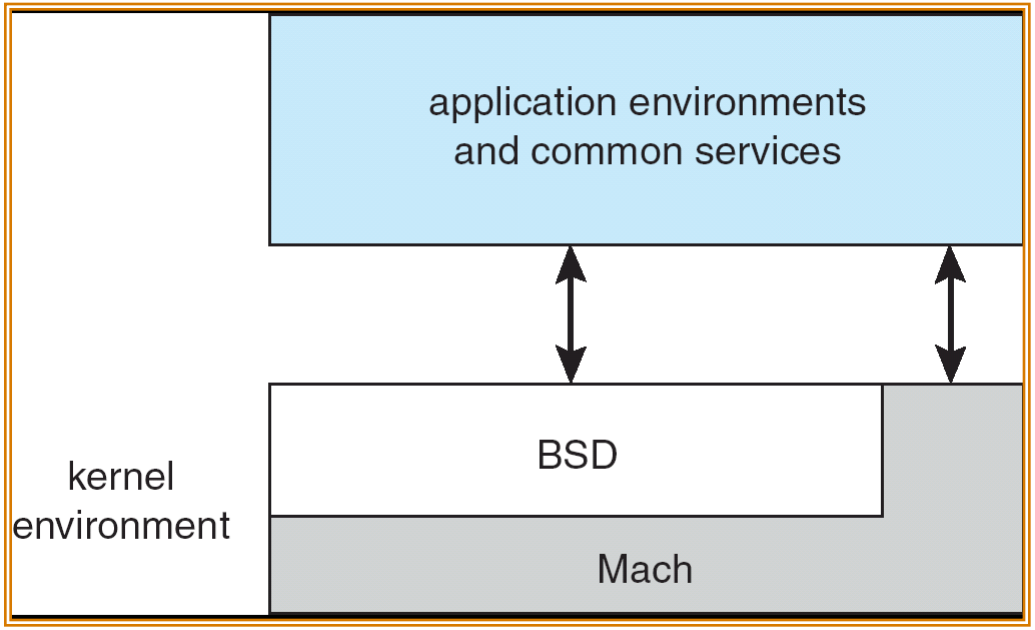
UNIX


(the users)
shells and commands
compilers and interpreters
system libraries
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
Kernel

Estructura MicroKernel

- Desplaza lo mas que puede del kernel al espacio de "*usuario*"
- Módulos (usuario) se comunican mediante mensajes (message passing)
- Beneficios:
  - Extensibilidad
  - Portabilidad a otras arquitecturas
  - Confiabilidad (menos código se ejecuta en modo kernel)
  - Seguridad
- Desventajas:
  - Costo de comunicación desde espacio de usuario al espacio de kernel

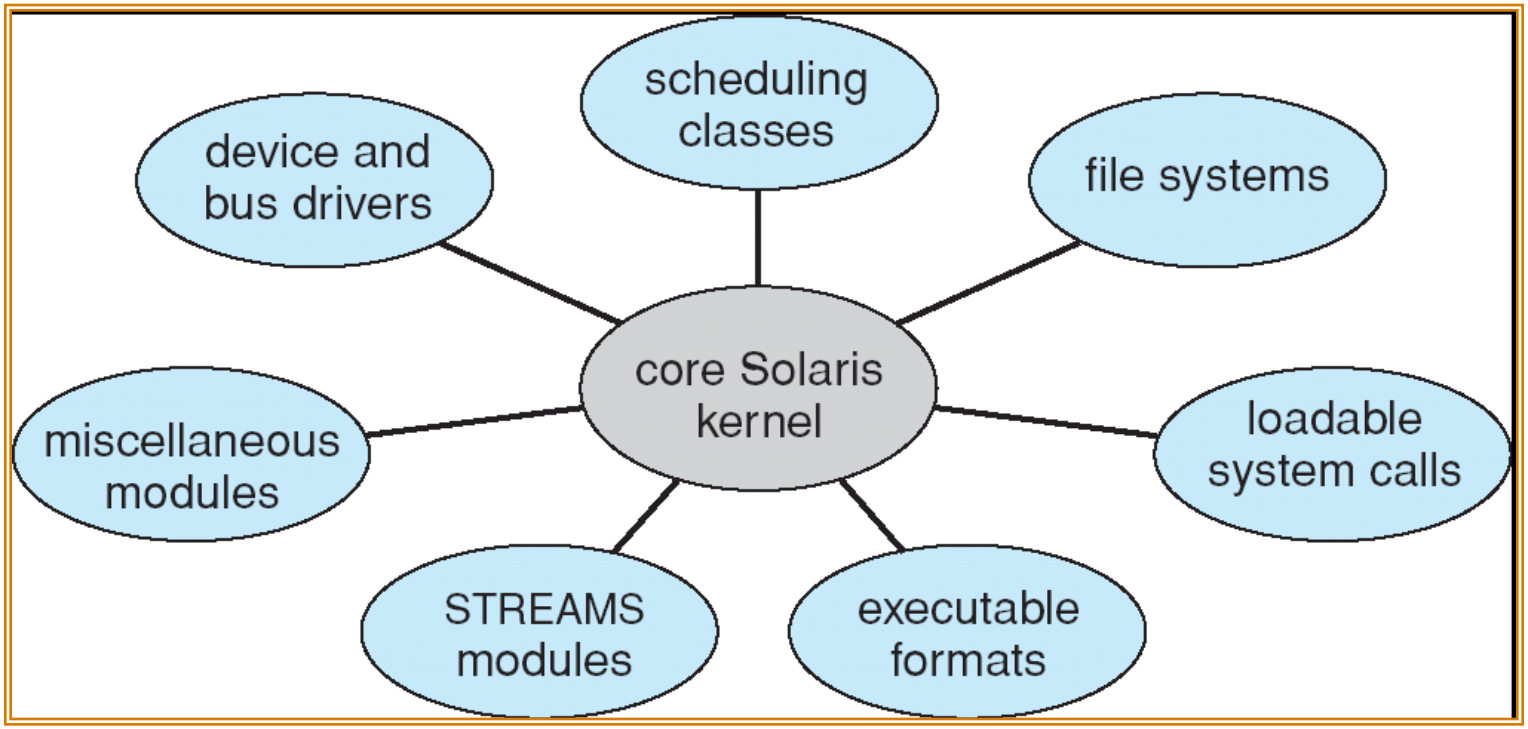
Estructura Mac OS X



Módulos

- Módulos del kernel
  - Desarrollados usando OOP
  - Componentes independientes
  - Comunicación mediante interfaces conocidas
  - Cargados a demanda
- En general, similar a layers, pero mas flexibles

Módulos en Solaris



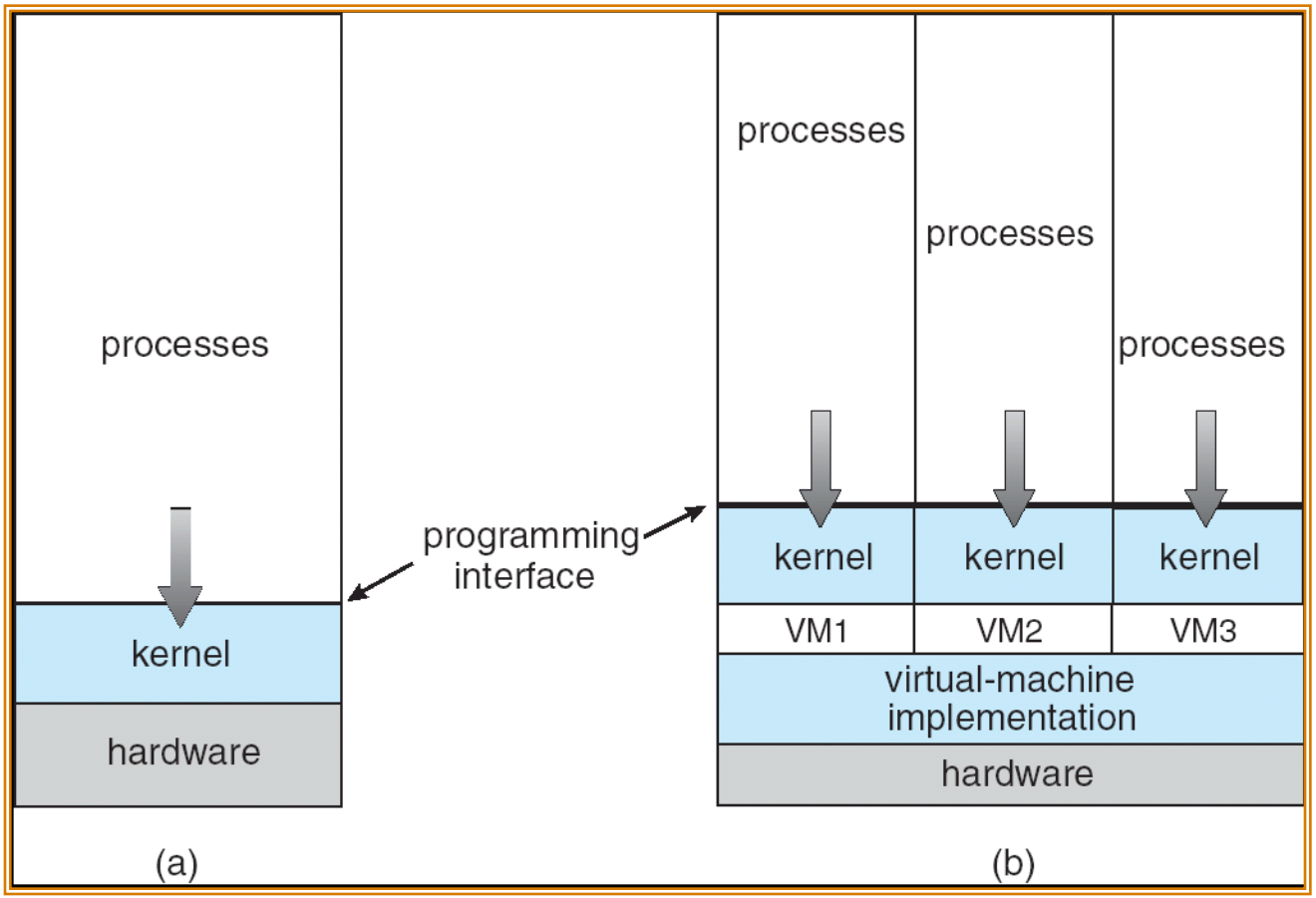
Máquinas Virtuales

- Una *máquina virtual* lleva el método de niveles a su conclusión lógica. Trata el hardware y el kernel del SO como si todo fuera hardware
- Proporciona una interfaz *identica* al hardware subyacente
- El SO crea la ilusión de procesos múltiples. Cada uno se ejecuta en su propio procesador con su propia memoria (virtual)

Máquinas Virtuales (Cont.)

- Los recursos de la computadora real son compartidos para crear la máquina virtual
  - El despachador de CPU puede crear la impresión de que los usuarios tienen su propio procesador
  - Un sistema de archivos y el spooling pueden proporcionar impresoras y lectores de dispositivos virtuales
  - La terminal con tiempo compartido de un usuario normal sirve como la consola de operación de la máquina virtual

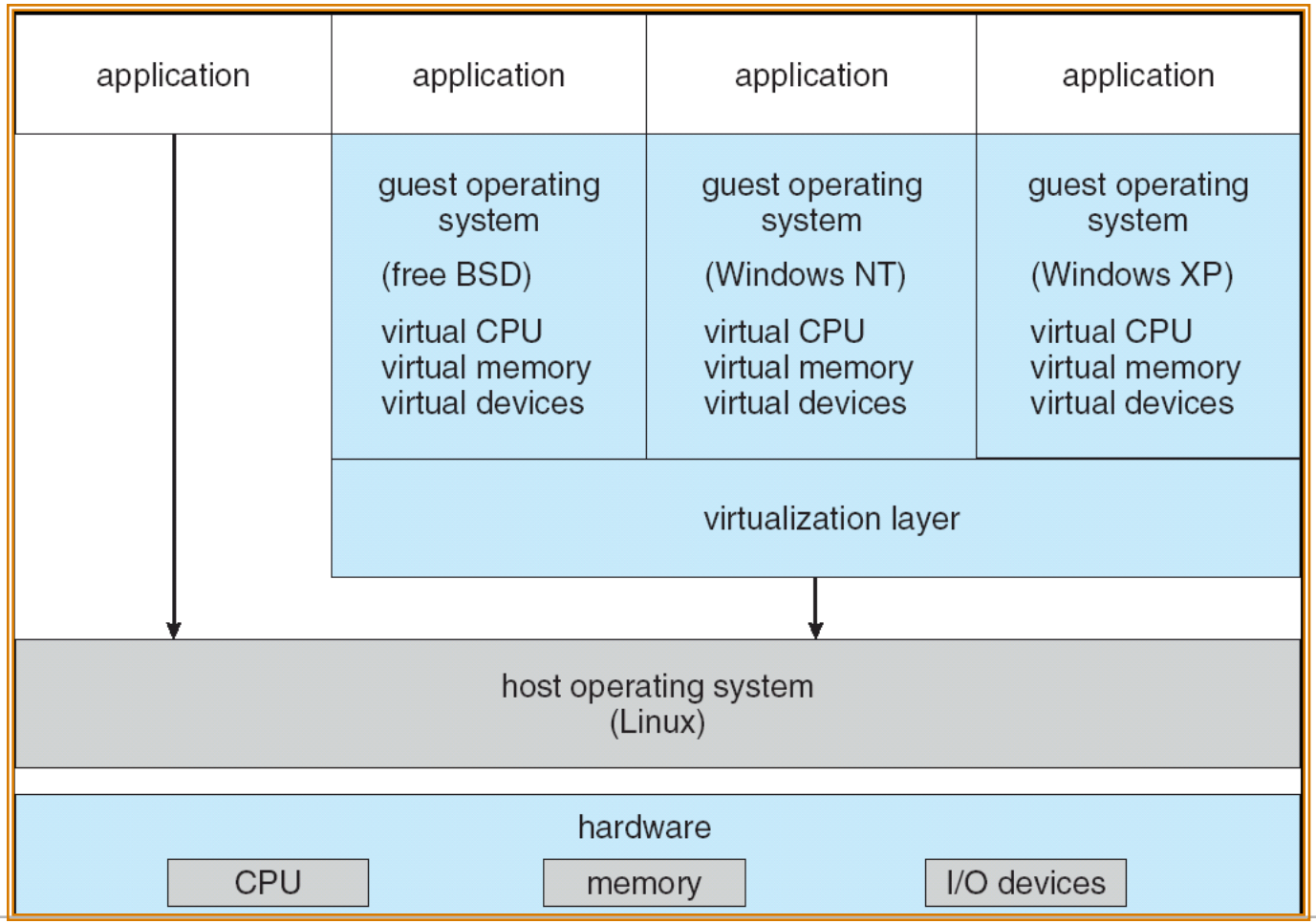
Máquinas Virtuales (Cont.)



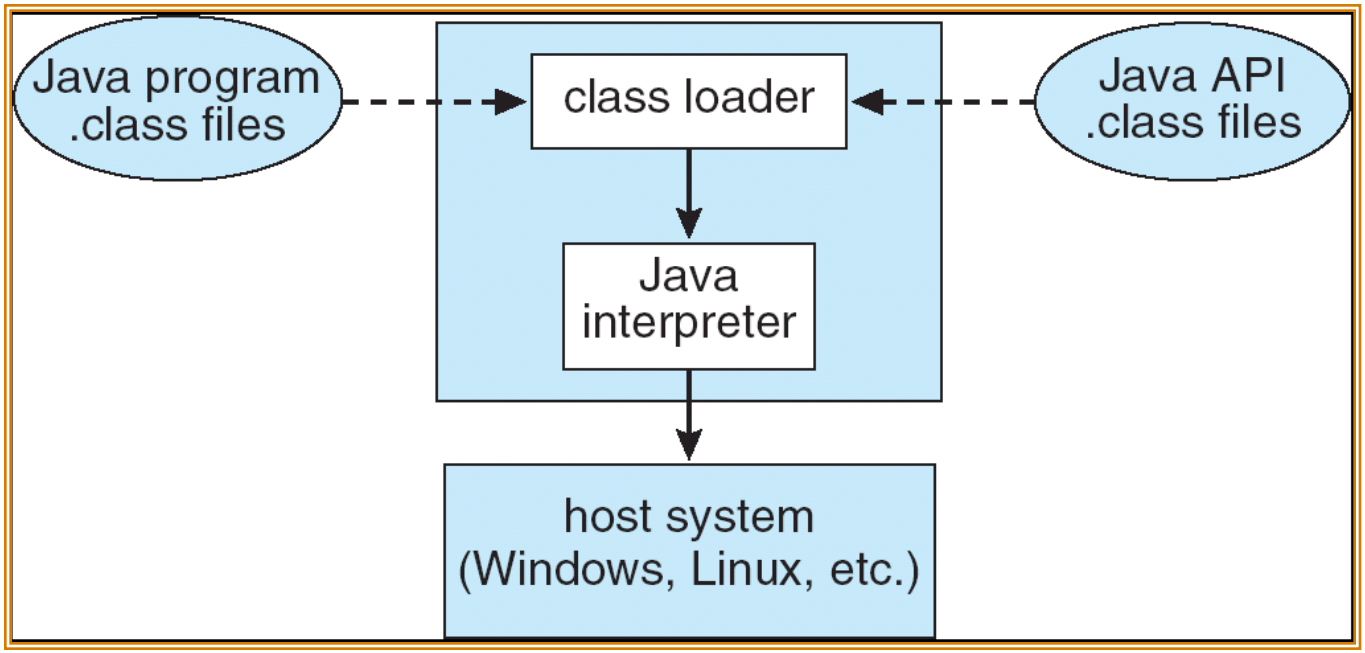
Máquinas Virtuales (Cont.)

- El concepto de máquina virtual proporciona protección completa de los recursos del sistema, dado que una máquina virtual está completamente aislada de las demás. Sin embargo, esta característica prohibe compartir recursos.
- Una máquina virtual es el vehículo perfecto para investigación y desarrollo en Sos. Reboots, etc., no interrumpen la operación normal de una computadora (compartida).
- El concepto de máquina virtual es difícil de implementar debido al esfuerzo requerido para proporcionar un duplicado *exacto* de la máquina subyacente.

Arquitectura de VMware



La Máquina Virtual de Java



Generación de SOs

- Los SOs se diseñan para correr en cualquier clase de máquina. El sistema debe estar configurado para cada sitio específico
- El programa SYSGEN obiene información de la configuración específica del hardware
- *Booting* – Iniciar una computadora cargando el kernel
- *Programa Bootstrap* – código almacenado en ROM capaz de localizar el kernel, cargarlo en memoria, y ejecutarlo

System Boot

- El SO debe estar disponible para que el hardware lo ejecute
  - El **bootstrap loader**, localiza al kernel, lo carga en memoria y lo ejecuta
  - En algunos casos, se lleva a cabo un proceso de dos niveles. Un **boot block** almacenado en una localidad fija (de disco), carga al bootstrap loader
  - Cuando se enciende la máquina, la ejecución comienza en una localidad fija de memoria
    - El Firmware almacena el código de boot