

# Tema 2

## Introducción a los Sistemas Operativos

Ingeniería técnica de Informática de  
Gestión 2009-2010

# Índice

- 1. Concepto de S. O. (Tanenbaum [P1-3, C1.1])
- 2. Historia y evolución (Tanenbaum [C1.2])
- 3. Tipos de Sistemas Operativos (Tanenbaum [C1.3])
- 4. Conceptos claves en S.O.
  - 4.1. Según el usuario de órdenes/aplicaciones (Silberschatz [C.3.4], Carretero [2.12])
  - 4.2. Según el usuario programador (Tanenbaum [C1.6,C.10.1.5])
  - 4.3. Según el diseñador/implementador (Silberschatz [C.3.1])
- 5. Carga y activación de un S.O. (Carretero [C2.2])
- 6. Arquitectura de Sistemas Operativos (Tanenbaum [C1.7])
- 7. Introducción a Unix/Linux (Tanenbaum [C10.1,10.2.1,10.2.2,10.2.5])
- 8. Introducción a Windows 2000 (Tanenbaum [11.1,11.3.1,11.3.3])

# Índice

## 1. Concepto de Sistema Operativo

## 2. Historia y evolución

2.1. 1ª generación (1945-1955): Válvulas y conexiones

2.2. 2ª generación (1955-1965): Transistores y sistemas de procesamiento de por lotes

2.3. 3ª generación (1965-1980): Circuitos integrados y multiprogramación

2.4. 4ª generación (1980-1990): Ordenadores personales

## 3. Tipos de Sistemas Operativos

3.1. S.O. de mainframe

3.2. S.O. de servidor

3.3. S.O. multiprocesador

3.4. S.O. para ordenadores personales

3.5. S.O. de tiempo real

3.6. S.O. integrados

3.7. S.O. de tarjeta inteligente

# 1. Concepto de S.O

- Los ordenadores vienen equipados con una capa de **software llamado S.O.** cuya labor es:
  - **administrar** todos los dispositivos del ordenador, ocultando sus peculiaridades
  - **proporcionar** al programador una interfaz de acceso sencilla para comunicarse con los dispositivos
- Unix, Linux, MS-DOS, Windows, FreeBSD, etc.
- El intérprete de órdenes, los sistemas de ventanas, los editores, los compiladores, etc., son **programas del sistema**, NO son el S.O.
- El S.O. se ejecuta en **modo núcleo**
- Los programas del sistema en **modo usuario**

# 1. Concepto de S.O (ii)

Básicamente hay dos visiones para definir un S.O.:

- **Máquina extendida o virtual:** (Perspectiva descendente)
  - **presenta** una **abstracción** del HW subyacente más sencilla y fácil de usar, ocultando sus peculiaridades
  - **presta** una variedad de **servicios** que los programas utilizan mediante instrucciones especiales, i.e., **llamadas al sistema**
- **Administrador o controlador de recursos:** (Perspectiva ascendente)
  - **administra** todos los elementos del ordenador
  - **reparte** ordenada y controladamente los elementos del sistema entre los programas que los solicitan

## 2.1. Historia y Evolución: Primera Generación

Históricamente unidos a la arquitectura de ordenadores

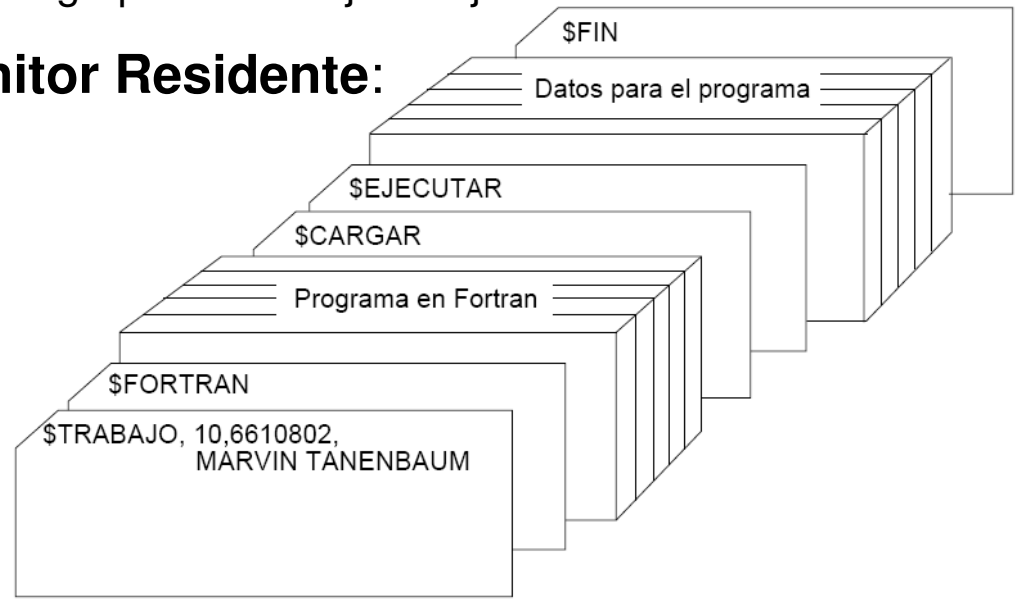
### Primera Generación (1945–1955): Válvulas y conexiones

- No  $\exists$  los S. O.
- **Interacción directa** )  $\Rightarrow$  grupo de personas diseñaba, construía, programaba, operaba y mantenía la máquina
- Tarjetas perforadas, instrucción a instrucción, lenguaje máquina
- Desarrollo lento de programas
- Desaprovechamiento de la máquina
- Código para controlar la E/S

## 2.2. Historia y Evolución: Segunda Generación

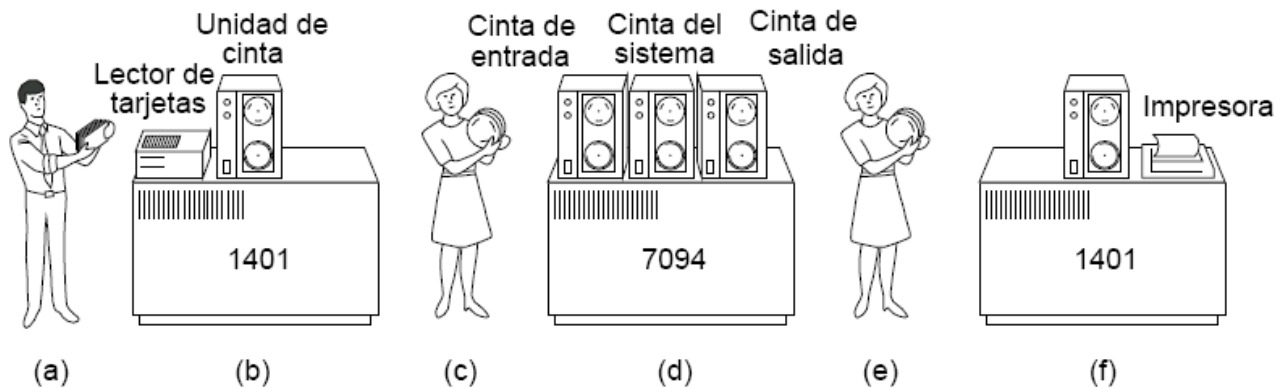
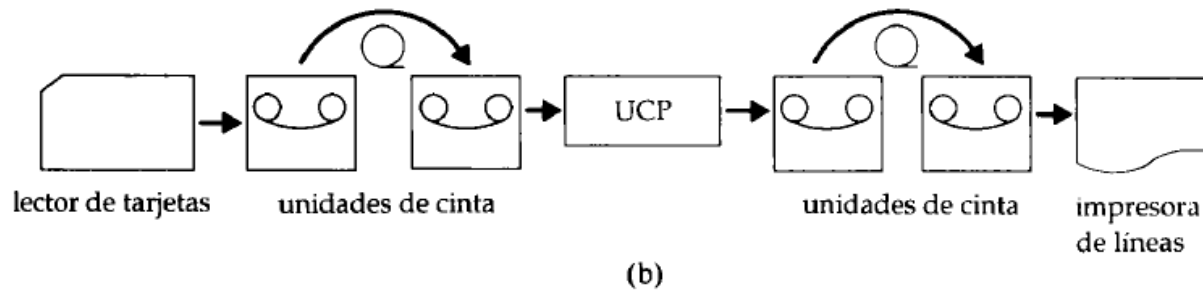
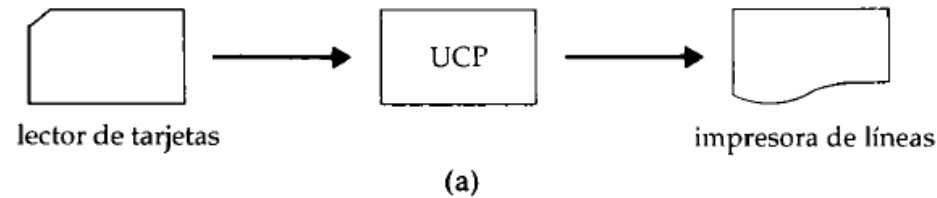
### Segunda Generación (1955–1965): Transistores y sistemas de procesamiento por lotes

- 1<sup>os</sup> Sistemas Operativos  $\Rightarrow$  transferir el control de un trabajo al sgte
- Librerías de funciones de E/S  $\Rightarrow$  programa no necesita controlar E/S
- **Monitor Residente**  $\Rightarrow$  interpreta las tarjetas de control
- **Procesamiento por lotes**  $\Rightarrow$  agrupando trabajos a ejecutar
- Datos para el programa **Monitor Residente**:
  - Intérprete de tarjetas
  - Cargador de programas
  - Controlador de E/S



## 2.2. Historia y Evolución: Segunda Generación (ii)

- Operación fuera de línea  $\Rightarrow$  ordenadores para cálculos y otros para E/S



## 2.2. Historia y Evolución: Segunda Generación (iii)

**Primeros intentos de solapar E/S con CPU:** Teniendo un único ordenador hay dos posibilidades, usar almacenamiento temporal con *buffers* o bien *Spoolers*

### **Buffers**

- Almacenamiento intermedio entre dispositivos de E/S y CPU, desde donde lee/escribe la CPU
- Cuando la CPU lee los datos del *buffer* pasa a trabajar con ellos
- A su vez, el dispositivo de E/S inicia la siguiente lectura
- Dispositivo y CPU trabajando a la vez
- Ideal si velocidad de CPU es  $\approx$  E/S

## 2.2. Historia y Evolución: Segunda Generación (iv)

### **Spoolers** (*Simultaneous Peripheral Operation On-Line*)

- Funcionamiento en el caso de entrada
  - La CPU solicita al lector la siguiente tarjeta y continúa su trabajo
  - Cuando el lector tiene los datos, interrumpe a la CPU, que copia los datos al disco duro y solicita la siguiente tarjeta
  - Cuando un proceso solicita los datos de la tarjeta, el S.O. los toma del disco y se los da al proceso
- Funcionamiento en caso de salida
  - La CPU escribe en disco la salida de un proceso
  - Al terminar, la CPU va mandando a la impresora los datos
- Se necesitan discos que permiten acceso aleatorio
- La CPU y los dispositivos de E/S están ocupados

## 2.3. Historia y Evolución: Tercera Generación

### Tercera Generación (1965–1980): Circuitos integrados y multiprogramación

- **Multiprogramación**
  - Varios trabajos en memoria: cuando uno espera para E/S, otro se puede ejecutar
  - Necesidad de planificación y protección
- **Tiempo compartido (multitarea)**
  - Variante del anterior
  - Cambio rápido entre tareas: uso interactivo
  - Más complejo
- Ejemplos de S.O.: OS/360, MULTICS, UNIX, etc.

# IBM/360 modelo 30



# IBM/360



## 2.4. Historia y Evolución: Cuarta Generación

### Cuarta Generación (1980–Actualidad): Ordenadores Personales

- Circuitos LSI y VLSI
- MS-DOS/Windows y Linux (PCs) vs. UNIX (*Workstations*)
- Sistemas Operativos de Red
  - Usuario consciente de la existencia de varios ordenadores
  - Cada máquina tiene su propio S.O.
  - Máquinas independientes, interactúan en momentos concretos
- Sistemas Operativos Distribuidos
  - Varios ordenadores
  - Imagen única del sistema
  - Transparencia de localización
  - Compartición de recursos, tolerancia a fallos, paralelismo, etc.

# 3. Tipos de Sistemas Operativos

- **S.O. de mainframe**

- Orientados al procesamiento de varios trabajos a la vez, que casi todos necesitan gran cantidad de E/S
- 3 tipos de servicios: por lotes, procesamiento de transacciones y tiempo compartido
- Ejemplo de S.O.: OS/390 descendiente del OS/360

- **S.O. de servidor**

- Los servidores pueden ser PCs muy potentes, estaciones de trabajo o incluso mainframes
- Dar servicio a múltiples usuarios a través de la red
- Permiten compartir recursos HW y SW
- Prestan servicios de impresión, de ficheros o de Web
- Ejemplo de S.O.: UNIX (Linux, FreeBSD, Solaris), Windows 2000

## 3. Tipos de Sistemas Operativos (i)

- **S.O. multiprocesador**

- S.O. para trabajar con computadoras paralelas, multicomputadoras o multiprocesadores
- Suelen ser S.O. de servidor, con funciones añadidas especiales para comunicación y conectividad
- Ejemplo: Linux (o AIX) en un IBM SP-2

- **S.O. para ordenadores personales**

- Su misión es presentar una buena interfaz a un único usuario
- Su principal uso  $\Rightarrow$  procesamiento de textos, hojas de cálculo, acceso a Internet, etc.
- Ejemplos de S.O. Windows, Linux, MacOS X (Basado en
- FreeBSD)

## 3. Tipos de Sistemas Operativos (ii)

### ■ S.O. de tiempo real

- Su parámetro clave es el **tiempo**
- S.O. de tiempo real **riguroso**:
  - es indispensable que la acción se efectúe en cierto momento, o en un intervalo
  - un ejemplo de su uso está en los procesos industriales controlando máquinas de producción, centrales nucleares, controladores de aviación, etc.
- S.O. de tiempo real **no riguroso** :
  - es aceptable no cumplir de vez en cuando un plazo, aunque esa imprecisión se tiene que ajustar a unos parámetros, como un porcentaje de fallo
  - un ejemplo de su uso son los sistemas de audio digital o multimedia, VxWorks y QNX

# 3. Tipos de Sistemas Operativos (iii)

- **S.O. integrados**

- Para computadoras de bolsillo (palm-top) y sistemas integrados
- Tienen características de S.O. en tiempo real con limitaciones de tamaño, memoria y consumo de electricidad
- Ejemplos: PalmOS y Windows CE (Consumer Electronic), Linux&QT-Embed

- **S.O. de tarjeta inteligente**

- Son los más pequeños, se ejecutan en tarjetas inteligentes del tamaño de una tarjeta de crédito que contienen una CPU
- Grandes limitaciones: potencia de procesamiento y memoria
- Realizan como mucho una o varias funciones
- Orientadas a Java → un intérprete de la Máquina Virtual de Java
- Los applets se descargan a la tarjeta y la JVM los interpreta
- Si hay varios applets → multiprogramación y planificación

# Índice

## 4. Conceptos claves de Sistemas Operativos

### 4.1. Según el usuario de órdenes/aplicaciones

4.1.1 Usuario

4.1.2 Sesión

4.1.3 Programa

4.1.4 Proceso

4.1.5 Fichero

4.1.6 Programas del sistema

4.1.7 Interfaz de usuario

### 4.2. Según el usuario programador

4.2.1 Llamadas al sistema

4.2.2 Estándar POSIX

4.2.3 API Win32 de Windows

### 4.3. Según el diseñador/implementador

4.3.1 Subsistema de gestión de procesos

4.3.2 Subsistema de gestión de memoria

4.3.3 Subsistema de gestión de E/S

4.3.4 Subsistema de gestión de ficheros

4.3.5 Subsistema de gestión de protección

# 4.1. Conceptos Claves de S.O.

## Según el usuario de órdenes/aplicaciones

- **Usuario** ⇒ Persona que trabaja en el sistema
- **Sesión** ⇒ Periodo de tiempo durante el cual un usuario interactúa con el sistema
- **Programa** ⇒ Código ejecutable. Concepto estático
- **Proceso** ⇒ Programa en ejecución. Concepto dinámico
- **Fichero** ⇒ Unidad lógica de almacenamiento
- **Programas del sistema** ⇒ Ofrecen un entorno más cómodo para el desarrollo y ejecución de programas
- **Interfaz de usuario** ⇒ Permite dar órdenes al sistema para realizar diversas operaciones

## 4.1.6. Programas del Sistema

- Suelen venir con el S.O. y dependen de él
- Normalmente realizan funciones básicas:
  - **Manipulación de ficheros** (crear, eliminar, ver, imprimir, etc.)
  - **Información de estado** (procesos, memoria, disco, etc.)
  - **Modificación de ficheros** (crear y modificar su contenido)
  - **Apoyo a lenguajes de programación** (compiladores, etc.)
  - **Comunicaciones** (correo, ftp, etc.)
  - **Aplicaciones** (editores de texto, de gráficos, etc.)
- El intérprete de órdenes o **shell** es un programa del sistema
- El S.O. no diferencia entre los programas del usuario y los del sistema

## 4.1.7. Interfaz de usuario (Carretero[2.12])

### Interfaces alfanuméricas

- Su modo de trabajo está basado en líneas de texto
- El usuario escribe la orden a ejecutar y sus parámetros
- El **intérprete de órdenes** es el módulo encargado de la interfaz ⇒ lee la orden, y ejecuta la acción especificada
- Existen dos tipos de intérpretes de órdenes:
  - **interno**: un único programa contiene el código para ejecutar todas las órdenes
  - **externo**: las órdenes no forman parte del intérprete, son programas externos
- En los sistemas reales puede existir una mezcla de las dos estrategias

## 4.1.7. Interfaces alfanuméricas

### Interfaz de órdenes **interno**

- Es más eficiente, pero puede llegar a ser muy grande, y posibles modificaciones o ampliaciones exigen cambiar el código del intérprete y recompilarlo
- Su funcionamiento es:
  - Lee la orden
  - Determina qué orden es, y salta a la parte del código correspondiente
  - Si no es una orden interna, ejecutará la aplicación en un nuevo proceso
  - Espera a que finalice la ejecución
- *command.com* de *MS-DOS* es un intérprete de órdenes interno

## 4.1.7. Interfaces alfanuméricas (i)

### Interfaz de órdenes **externo**

- La interfaz de usuario está compuesta por el intérprete y cada uno de los programas del sistema
- Su funcionamiento es:
  - Lee la orden
  - Crea un nuevo proceso que ejecutará esa orden
  - Espera a que termine la ejecución
- Es menos eficiente, y las modificaciones o inclusiones de nuevas órdenes sólo implican añadir nuevos programas
- Los *shells* de *UNIX* son externos, aunque algunas órdenes están implementadas como internas (p. ej. `cd`, `echo`, `expr`, `test`)

## 4.1.7. Interfaz de usuario

### Interfaces gráficas, GUI (*Graphical User Interface*)

- El objetivo es presentar a los usuarios una visión sencilla e intuitiva del sistema, ocultando su complejidad
- Están basadas en **ventanas** que permiten trabajar simultáneamente con varias actividades
- Se utilizan **iconos** y **menús** para representar los recursos y poder realizar operaciones sobre los mismos
- El **ratón** permite interactuar con estos elementos
- Para usuarios avanzados y para agilizar el trabajo proporcionan la posibilidad de realizar las mismas operaciones mediante una combinación de teclas

## 4.1.7. Interfaces gráficas

- La estructura interna está formada por un conjunto de programas, los cuales trabajan conjuntamente para realizar las peticiones del usuario, usando los servicios del sistema
- Gestor de ventanas para mantener el estado de las mismas y permitir su manipulación
- Administrador de programas para arrancar aplicaciones
- Gestor de archivos para manipular ficheros y directorios
- Herramientas de configuración de la propia interfaz y del entorno
- Ideal si también incluye alguna otra interfaz “programática” a nivel de *scripts* (importante separar la funcionalidad de la presentación)

## 4.2. Conceptos Claves de S.O.

### Según el usuario programador

- **Llamadas al sistema**

- Definen Interfaz entre el S.O. y los programas de usuario
- Dependen mucho de la máquina y en ocasiones están en código ensamblador
- Lenguajes de alto nivel tienen *librerías de procedimientos* que permiten hacer llamadas al sistema
- Un proceso de usuario necesita un servicio del S.O. ⇒ llamada al sistema ⇒ control al S.O.
- Entonces el S.O. realiza las siguientes tareas:
  - determina qué quiere el proceso invocador, examinando los parámetros
  - ejecutará la llamada al sistema
  - devuelve el control a la instrucción que está después de la llamada al sistema
- Llamada a un procedimiento especial que se ejecuta en **modo núcleo**

## Ejemplo: llamada read en OSO (servicio 01 de int 22h)

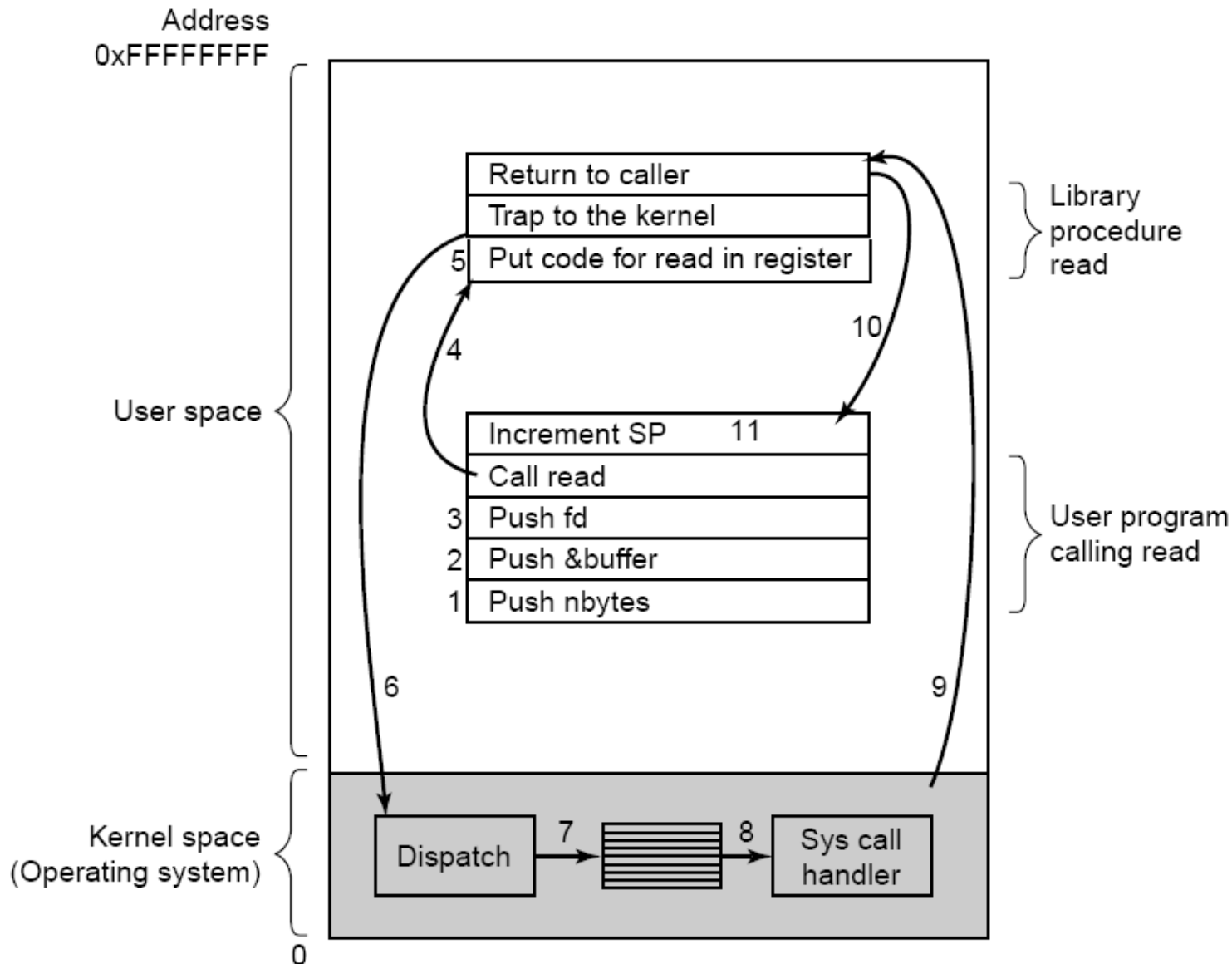
int read(int fd, void far \* buffer, unsigned int contador)

- Descripción: lee de un fichero.
- Servicio: **AH=01h**.
- Entrada: descriptor del fichero a leer (**DX**), buffer (**ES:BX**) y contador (**CX**).
- Salida: en AX, el número de bytes leídos o -1 si se produjo error. Si ya se llegó al final del fichero devuelve 0.

```
int read(int fd, char far * buffer, int count)
{
    asm {
        mov ah, 1 ; recoge de la pila los parámetros y los deja en
registros
        mov cx, count
        mov dx, fd
        les bx, dword ptr buffer
        int 22h; salto al kernel; instrucción de TRAP en OSO
    }
}
```

## 4.2.1. Llamadas al sistema

- `cuenta=read(fd, buffer, nbytes);`



## 4.2.2. Según el usuario programador

- **Estándar POSIX (1003.1)** (Tanenbaum [C.10.1.5])
  - *Portable Operating System + UNIX*
  - Realizado bajo la IEEE, participaron la industria, las universidades y el gobierno
  - Define el conjunto de procedimientos de librería que debe proporcionar todo sistema UNIX que cumpla con la norma
  - Casi todos son llamadas al sistema, pero algunos se pueden implementar fuera del núcleo
  - Programa que sólo use procedimientos definidos en POSIX ) se ejecutará en cualquier sistema UNIX
  - open, read, fork, etc.
  - Documentos relacionados estandarizan procesos ligeros, seguridad, shells y utilidades, servicios en tiempo real, etc.

## 4.2.2. POSIX: Algunas llamas

### Process management

| Call                                  | Description                                    |
|---------------------------------------|--|
| pid = fork( )                         | Create a child process identical to the parent |
| pid = waitpid(pid, &statloc, options) | Wait for a child to terminate                  |
| s = execve(name, argv, environp)      | Replace a process' core image                  |
| exit(status)                          | Terminate process execution and return status  |

### File management

| Call                                 | Description                              |
|--------------------------------------|--|
| fd = open(file, how, ...)            | Open a file for reading, writing or both |
| s = close(fd)                        | Close an open file                       |
| n = read(fd, buffer, nbytes)         | Read data from a file into a buffer      |
| n = write(fd, buffer, nbytes)        | Write data from a buffer into a file     |
| position = lseek(fd, offset, whence) | Move the file pointer                    |
| s = stat(name, &buf)                 | Get a file's status information          |

### Directory and file system management

| Call                           | Description                                  |
|--------------------------------|--|
| s = mkdir(name, mode)          | Create a new directory                       |
| s = rmdir(name)                | Remove an empty directory                    |
| s = link(name1, name2)         | Create a new entry, name2, pointing to name1 |
| s = unlink(name)               | Remove a directory entry                     |
| s = mount(special, name, flag) | Mount a file system                          |
| s = umount(special)            | Unmount a file system                        |

### Miscellaneous

| Call                     | Description                             |
|--------------------------|---|
| s = chdir(dirname)       | Change the working directory            |
| s = chmod(name, mode)    | Change a file's protection bits         |
| s = kill(pid, signal)    | Send a signal to a process              |
| seconds = time(&seconds) | Get the elapsed time since Jan. 1, 1970 |

## 4.2.3. Según el usuario Programador

- **API Win32 de Windows** (*Application Programming Interface*)

- Programa Windows ⇒ controlado por eventos
- Programa Principal ⇒ espera evento ⇒ invoca un procedimiento para procesarlo, actualizar la ventana, y el estado interno del programa
- Eventos ⇒ pulsación de una tecla, movimiento o clic del ratón, inserción de un disquete, etc.

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

- Las llamadas al sistema y las librerías de procedimientos están desacopladas

## 4.2.3. API Win 32

- Continúa. . .

- API Win32  $\Rightarrow$  cjto. de procedimientos para pedir servicios al S.O.
- Se reconoce parcialmente en todas las versiones de Windows (desde Windows 95)
- Al desacoplar la interfaz de las llamadas al sistema, Microsoft se guarda la posibilidad de modificar las llamadas al sistema, sin inutilizar los programas existentes
- La API la forman millares de funciones, algunas provocan llamadas al sistema, otras no (se ejecutan en modo usuario)
- Difícil saber qué es una llamada al sistema o simplemente una llamada a un procedimiento de librería en el espacio de usuario
- Hablaremos de la API Win32, no de llamadas al sistema propiamente dichas
- También maneja muchas características de la GUI: ventanas, figuras geométricas, texto, tipos de letra, menús, etc.

## 4.2.3. APIWin32

Llamadas de API  $\approx$  llamadas al sistema de POSIX

| UNIX    | Win32               | Description  |
|---------|---------------------|--|
| fork    | CreateProcess       | Create a new process                               |
| waitpid | WaitForSingleObject | Can wait for a process to exit                     |
| execve  | (none)              | CreateProcess = fork + execve                      |
| exit    | ExitProcess         | Terminate execution                                |
| open    | CreateFile          | Create a file or open an existing file             |
| close   | CloseHandle         | Close a file                                       |
| read    | ReadFile            | Read data from a file                              |
| write   | WriteFile           | Write data to a file                               |
| lseek   | SetFilePointer      | Move the file pointer                              |
| stat    | GetFileAttributesEx | Get various file attributes                        |
| mkdir   | CreateDirectory     | Create a new directory                             |
| rmdir   | RemoveDirectory     | Remove an empty directory                          |
| link    | (none)              | Win32 does not support links                       |
| unlink  | DeleteFile          | Destroy an existing file                           |
| mount   | (none)              | Win32 does not support mount                       |
| umount  | (none)              | Win32 does not support mount                       |
| chdir   | SetCurrentDirectory | Change the current working directory               |
| chmod   | (none)              | Win32 does not support security (although NT does) |
| kill    | (none)              | Win32 does not support signals                     |
| time    | GetLocalTime        | Get the current time                               |

# 4.3. Conceptos Claves de S.O.

## Según el diseñador/implementador

- Subsistema de gestión de procesos
  - Proceso  $\Rightarrow$  **programa en ejecución**
  - Trabajo por lotes, programa de usuario, tarea del sistema, etc.
  - Necesita de recursos para su ejecución: tiempo de CPU, memoria, archivos, dispositivos de E/S, etc.
  - Programa  $\Rightarrow$  entidad pasiva
  - Proceso  $\Rightarrow$  entidad activa, tiene un contador de programa que dice cuál es la siguiente instrucción a ejecutar
  - El proceso es la unidad de trabajo del sistema
  - Actividades del S.O.:
    - Crear y eliminar procesos de usuario y de sistema
    - Suspende y reanuda la ejecución de procesos
    - Comunicación y sincronización entre procesos

## 4.3. Según el diseñador/implementador

- Subsistema de gestión de la memoria principal
  - Principal almacenamiento de datos de acceso rápido y compartido por la CPU y los dispositivos de E/S
  - Los procesos deben estar total o parcialmente en memoria
    - Entran y salen de memoria nuevos procesos
    - Utilizan memoria dinámica que reservan y liberan
    - Áreas internas al S.O. como DMA y contabilidad
  - Es necesario mantener varios procesos en memoria para mejorar el uso de la CPU
  - **¿Qué esquema de administración de memoria utilizar?**
  - Actividades del S.O.:
    - Llevar control de las zonas de memoria ocupadas y por quién, y de las zonas libres
    - Decidir qué procesos se cargan en memoria
    - Asignar y recuperar espacio en memoria

## 4.3. Según el diseñador/implementador (i)

- Subsistema de gestión de E/S
  - Ocultar las particularidades del hardware y que el usuario no las perciba
  - Ofrecer una interfaz homogénea
  - En UNIX se consigue mediante el subsistema de E/S:
    - Un componente de gestión de memoria que incluye el uso de *buffers*, *caché*
    - Interfaz general (y uniforme) con los manejadores de dispositivos
    - Manejadores para dispositivos hardware específicos
  - Sólo el manejador de dispositivo conoce sus peculiaridades
  - Por ejemplo, para la gestión del almacenamiento secundario:
    - Administración del espacio libre
    - Asignación de almacenamiento
    - Planificación del disco

## 4.3. Según el diseñador/implementador (ii)

- Subsistema de gestión de ficheros
  - Es el componente más visible del S.O.
  - Disquetes, discos magnéticos, discos ópticos, etc.
  - Presentar una perspectiva lógica uniforme de almacenamiento
  - de información, abstrayéndose de las propiedades del dispositivo
  - Fichero  $\Rightarrow$  unidad lógica de almacenamiento
  - Fichero  $\Rightarrow$  colección de información relacionada (numéricos, alfabéticos, alfanuméricos, con formato libre o rígido, etc.)
  - Directorios  $\Rightarrow$  organizar ficheros para facilitar su uso
  - Actividades del S.O.:
    - Crear y eliminar ficheros y directorios
    - Permitir la manipulación de ficheros y directorios
    - Correspondencia entre ficheros y almacenamiento secundario

## 4.3. Según el diseñador/implementador (iii)

- Subsistema de protección
  - **Protección**  $\Rightarrow$  mecanismo para controlar el acceso de los procesos o usuarios a los recursos definidos por un sistema de computación: ficheros, impresoras, procesos, etc.
  - Especificar los controles que se impondrán y cómo ponerlos
  - Los distintos procesos deben protegerse unos de otros
  - Ficheros, segmentos de memoria, CPU, etc.  $\Rightarrow$  sólo deben ser usados por los procesos que han obtenido autorización
  - Ya hemos visto a nivel HW
    - Modo núcleo/usuario  $\Rightarrow$  controlar los dispositivos HW
    - Registros base y límite  $\Rightarrow$  controlar la memoria
    - Cronómetros  $\Rightarrow$  controlar la CPU

# Índice

## 5. Carga y Activación de un S.O.

5.1. Arranque hardware

5.2. Ubicación del Sistema Operativo

5.3. Arranque del Sistema Operativo

## 6. Arquitecturas de Sistemas Operativos

6.1. Sistemas monolíticos

6.2. Sistemas en capas

6.3. Máquinas virtuales

6.4. Exokernels

6.5. Modelo cliente/servidor

6.6. Modelo OO o de objetos distribuidos y componentes

# 5. Carga y Activación de un S.O.

El arranque de un ordenador actual tiene 2 fases:

- arranque hardware
- arranque del S.O.

Bajo el control del  
iniciador ROM

{ Test del hardware  
Carga en memoria del cargador del SO

Bajo el control del  
cargador (*boot*) del SO

{ Carga en memoria componentes del SO

Inicialización bajo el  
control de la parte  
residente del SO

{ Test del sistema de ficheros  
Creación de estructuras de datos internas  
Completa la carga del SO residente  
Creación de procesos *login*

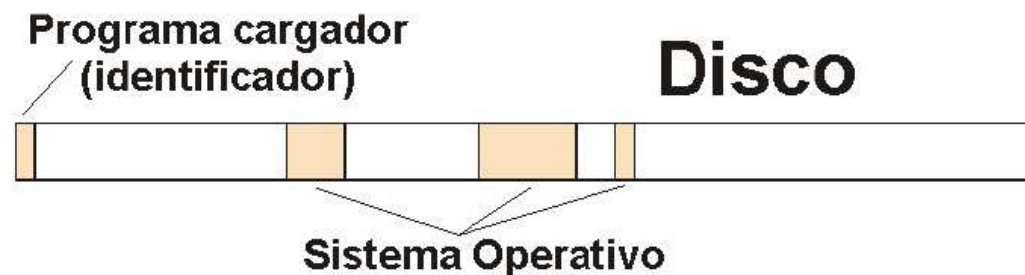
Se entra en la fase normal de funcionamiento del SO

# 5.1. Arranque hardware

- **Iniciador ROM**  $\Rightarrow$  Programa de arranque disponible en la ROM
- Al arrancar el ordenador  $\Rightarrow$  Señal eléctrica  $\Rightarrow$  Carga valores predefinidos en los registros
- Contador del programa  $\Rightarrow$  dirección de inicio del iniciador ROM
- El iniciador ROM realiza tres funciones:
  - 1. Comprueba el sistema, detectando sus características y comprobando su funcionamiento
  - 2. Lee y almacena en memoria el programa cargador del S.O.
  - 3. Pasa el control al cargador del S.O., saltando a la dirección de memoria donde lo ha almacenado
- PC's  $\Rightarrow$  la ROM tiene también un SW de E/S, la **BIOS**

## 5.2. Ubicación del Sistema Operativo

- S.O. está almacenado en una unidad de disco
- Programa cargador (o *boot*) del S.O.  $\Rightarrow$  primeros sectores del disco y con un tamaño prefijado
- *Master Boot Record* o *Volume Boot Record*
- Programa iniciador de la ROM y S.O. tienen un acuerdo sobre el cargador del S.O.:
  - ubicación
  - dirección de arranque
  - tamaño
- Esto permite que el iniciador pueda soportar varios S.O.



## 5.3. Arranque del Sistema Operativo

- Programa cargador del S.O.  $\Rightarrow$  trae a memoria algunos componentes del S.O.
- Empieza la fase de iniciación del S.O.:
  - Comprobación del sistema
  - Se establecen estructuras internas del S.O.: tabla de procesos, tabla de memoria, de E/S
  - Se carga la parte del S.O. que va a estar siempre en memoria, **sistema operativo residente**
  - Se lanzan los procesos auxiliares y *demonios* (impresión, red, etc.)
  - Se crea un proceso de inicio o *login*, por cada terminal

# 6.1. Arquitecturas de S.O.

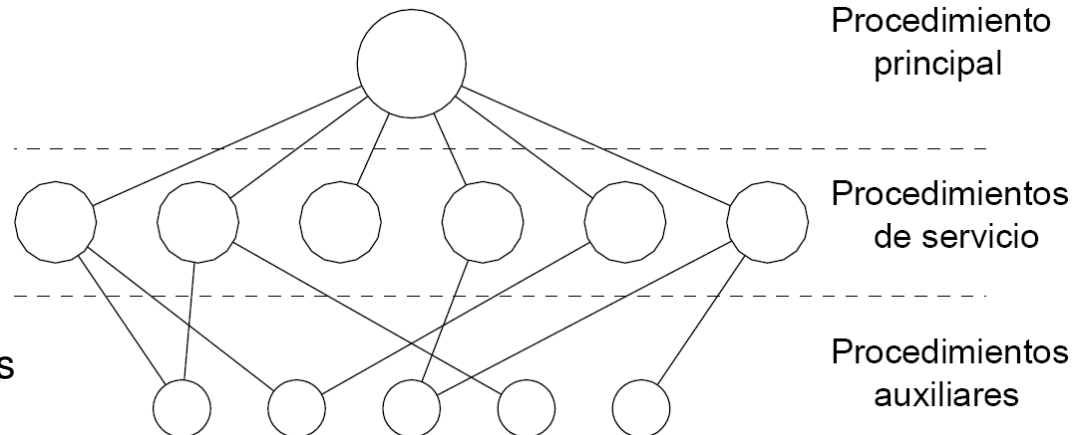
## Sistemas monolíticos (“El Gran Embrollo”)

- Colección de procedimientos sin estructura
- Interfaz de procedimientos bien definida
- No hay ocultamiento de información
- Llamadas al sistema para solicitar servicios al S.O.
- MS-DOS, UNIX, Windows
- Estructura básica:

Progr. ppal. ⇒ procedimiento del servicio solicitado

Procedimientos de servicio ejecutan llamadas al sistema

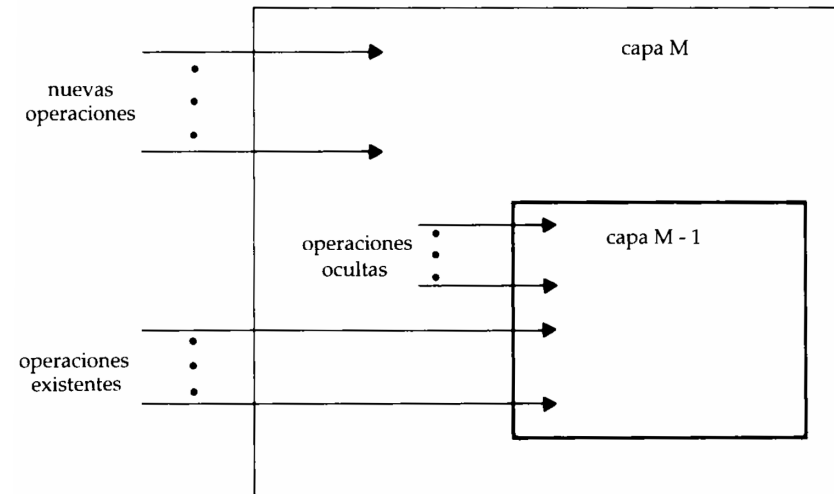
Procedimientos utilitarios apoyan a los procedimientos de servicio



## 6.2. Arquitecturas de S.O. (i)

### Sistemas en Capas

- Jerarquía de capas
- Ventajas
  - Modularidad
  - Ocultación de la información
  - Verificación por capas
- Inconvenientes
  - Difícil diseño de las capas
  - Dependencias múltiples
  - ¿Dónde se pone esto?
- THE, VENUS
- MULTICS (anillos concéntricos)



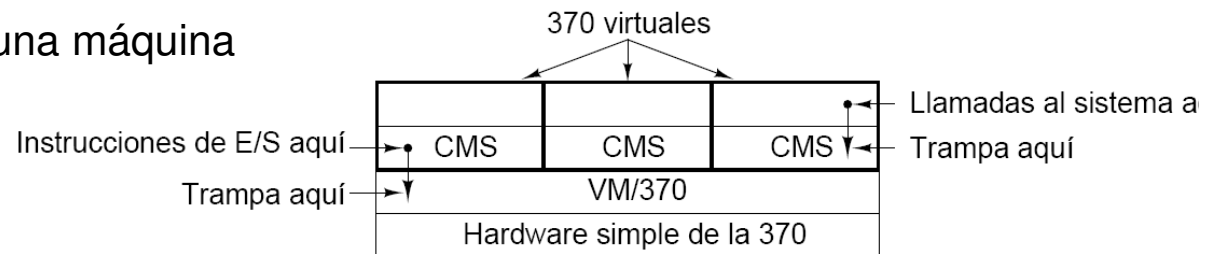
| Layer | Function                                  |
|-------|---|
| 5     | The operator                              |
| 4     | User programs                             |
| 3     | Input/output management                   |
| 2     | Operator-process communication            |
| 1     | Memory and drum management                |
| 0     | Processor allocation and multiprogramming |

Estructura del S.O. THE

# 6.3. Arquitecturas de S.O. (ii)

## Máquinas virtuales

- El S.O. no añade funcionalidad nueva
- El **monitor de máquina virtual** se ejecuta sobre el HW desnudo, realiza multiprogramación y proporciona varias máquinas virtuales a la capa superior
- Varias Máquinas Virtuales sobre una máquina física
- Es complejo: duplicado idéntico de la máquina física
- CMS, Modo 8086 virtual de Windows, VMWARE, JVM
- Ventajas
  - Protección total de los recursos
  - Desarrollo de nuevos S.O.
  - Distintos S.O. sobre una máquina (VMWARE)



## 6.4. Arquitecturas de S.O. (iii)

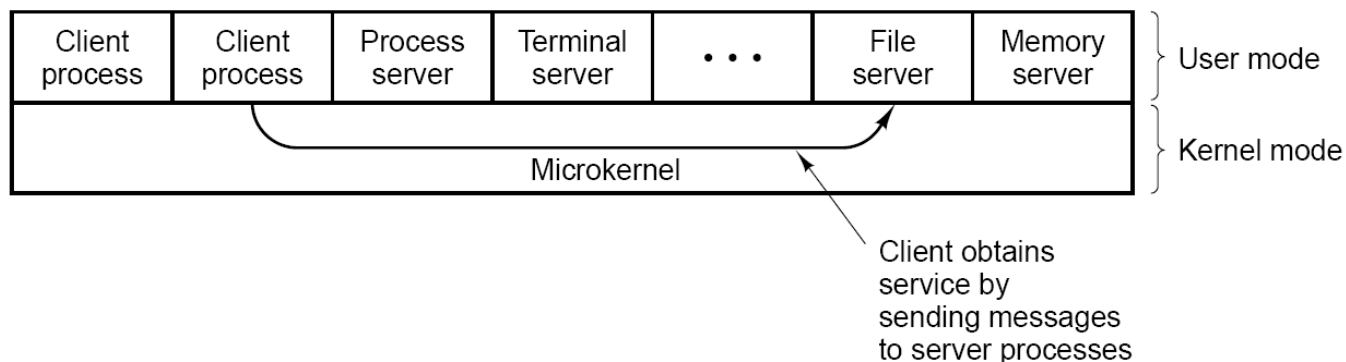
### Exokernels

- Presentan un clon exacto de la computadora real, pero con un subconjunto de los recursos de la misma
- Exokernel:
  - Se ejecuta, en modo núcleo, en la capa más baja
  - Asigna recursos a las máquinas virtuales
  - Controlar los intentos de uso de recursos, evitando interferencias
  - Cada máquina virtual a nivel usuario puede ejecutar su propio S.O. pero limitado a los recursos que se le asignan
- Ahorra una capa de mapeo o correspondencia
- Separa la multiprogramación (en el exokernel) y el código del S.O. del usuario (en el espacio de usuario)

# 6.5. Arquitecturas de S.O. (iv)

## Modelo Cliente/Servidor

- Proceso **Cliente**: Solicita servicios
- Proceso **Servidor**: Provee servicios
- El **núcleo** se convierte en un microkernel:
  - Maneja la comunicación entre clientes y servidores
  - Servidores en modo usuario:
    - Protección: no acceso directo al hardware ¿entonces?
    - Bien algunos servidores en modo núcleo (incluidos en núcleo)
    - Bien envían mensajes especiales que el kernel reconoce
  - Óptimo para **sistemas distribuidos** (extensible, escalable, etc.)



# Índice

## 7. Introducción a UNIX/Linux

### 7.1. Historia de UNIX/Linux

### 7.2. Estructura de UNIX/Linux

#### 7.2.1 Metas de diseño de UNIX/Linux

#### 7.2.2 Interfaz de UNIX/LINUX

#### 7.2.3 Características de Linux

#### 7.2.4 Kernel Linux

## 8. Introducción a Windows 2000

### 8.1. Historia de Windows 2000

### 8.2. Estructura de Windows 2000

#### 8.2.1 Capa de abstracción hardware, HAL

#### 8.2.2 Capa del Kernel

#### 8.2.3 El ejecutivo

#### 8.2.4 Manejadores de dispositivos

#### 8.2.5 Subsistema de entorno

# 7. Introducción a UNIX/Linux

## Historia de UNIX/Linux

- Ken Thompson, (Bell Labs), escribió un austero MULTICS (en ensamblador) para una PDP-7 ⇒ **UNICS** (*Servicio de Información y Cómputo Uniplexado*) ⇒ UNIX
- Thompson + Dennis Ritchie + su dpto. ⇒ **UNIX** a una PDP-11
- Deciden escribir UNIX en un lenguaje de alto nivel ) evitar dependencia con el hardware de la máquina:
  - Thompson diseñó el lenguaje **B**, e intenta escribir UNIX en B, pero el intento fue infructuoso
  - Ritchie diseñó el lenguaje **C**, (basado en B), para el que escribió un buen compilador
  - Thompson y Ritchie escribieron UNIX en C

# 7.1 Historia de UNIX/Linux

- Bell Labs cedió UNIX bajo licencia a las universidades
- Universidades: PDP-11 + código UNIX  $\Rightarrow$  auge UNIX
- Versiones 6 y 7. La 7  $\Rightarrow$  1ª versión portable, PDP-11 e Interdata 8/32, 18.800 líneas de C y 2.100 de ensamblador
- **UNIX Portable:**
  - UNIX a Interdata 8/32  $\Rightarrow$  no fue una tarea sencilla
  - Se diseñó e implementó un **compilador portable de C**
  - Trabajo para Interdata  $\Rightarrow$  trabajo con redes en UNIX
  - Pronto se trasladó a la VAX y a otras computadoras
  - AT&T  $\Rightarrow$  System III  $\Rightarrow$  System V
  - Se vendió el código a Santa Cruz Operation en 1995
  - Todas las compañías importantes tenían su licencia

# 7.1 Historia de UNIX/Linux (i)

## UNIX de Berkeley

- Universidad de California en Berkeley adquirió UNIX Versión 6
- **1BSD** (*First Berkely Software Distribution*) ⇒ versión mejorada para la PDP-11
- Versiones: **2BSD** para PDP-11, **3BSD** y **4BSD** para VAX
- En la versión **4BSD** incluyen un gran número de mejoras:
  - Memoria virtual y paginación
  - Nombres de archivo de más de 14 caracteres
  - Modificación del sistema de ficheros haciéndolo más rápido
  - Manejo de señales más fiable
  - Conectividad de redes, TCP/IP ) estándar de facto en UNIX
  - *vi*, *csh*, compiladores de *Pascal* y *Lisp*, . . . (programas del sistema)
- BSD ⇒ ámbito académico, y algunas compañías (Sun y DEC)

# 7.1. Historia de UNIX/Linux (ii)

## UNIX estándar: POSIX

- 1980 ⇒ dos versiones de UNIX distintas e incompatibles: **4.3BSD** y **System V Release 3**
- Poco éxito comercial de UNIX ⇒ difícil hacer software que funcione en todos los sistemas UNIX
- Nace el estándar POSIX ⇒ Intersección de System V y BSD + cosas nuevas (muy parecido a Versión 7)
- IBM + DEC + HP + otros ⇒ OSF (*Open Software Foundation*) ⇒ sistema que se ajustaba al estándar + sistema de ventanas
- (X11), interfaz gráfica (MOTIF), computación distribuida (DCE), . . .
- AT&T ⇒ UI (*UNIX Internacional*) ⇒ versión de UNIX basada en System V
- Mercado apostó por System V, OSF fue desapareciendo

# 7.1. Historia de UNIX/Linux (iii)

## • MINIX

- Tanenbaum escribió un nuevo sistema UNIX, pequeño, con el código disponible, y para finés educativos
- Salió en 1987 y era casi equivalente al UNIX Versión 7
- Diseño de **microkernel**
- Sistema de ficheros y administración de memoria  $\Rightarrow$  procesos de usuario
- Manejadores de dispositivos  $\Rightarrow$  en el kernel
- Se convirtió en un objeto de culto: grupo de noticias, usuarios aportando código, . . .
- En 1997 aparece la Versión 2.0 de Minix, que incluye conectividad de redes

## 7.1. Historia de UNIX/Linux (iv)

### • Linux

- Linus Torvalds, estudiante finlandés, escribió otro clon de UNIX: **Linux**. La primera versión, la 0.01, salió en 1991
- Sistema con algunas ideas prestadas de MINIX, inicialmente se desarrolló en una máquina MINIX
- Diseño **monolítico** del kernel (no microkernel como MINIX)
- 1994 sale la versión 1.0:
  - Nuevo sistema de ficheros
  - Archivos con correspondencia en memoria
  - Conectividad de redes compatible con BSD (sockets y TCP/IP)
  - Muchos manejadores nuevos
- 1996 sale la versión 2.0:
  - Manejo de arquitecturas de 64bits
  - Multiprogramación simétrica
  - Nuevos protocolos de redes, . . .

## 7.1. Historia de UNIX/Linux (v)

- Linux continúa. . .
  - Mucho software de UNIX se pasó a Linux: programas de sistema, X Windows, software de redes, etc.
  - Aparecen GNOME y KDE dos GUIs distintas
  - Linux es **Software Libre**, licencia GPL (*Licencia Pública GNU*):
    - El código fuente o binario se puede usar, copiar, modificar y redistribuir con libertad
    - Las modificaciones del Kernel de Linux han de venderse o redistribuirse siempre acompañadas del código fuente
  - Torvalds mantiene el control sobre el Kernel
  - Muchos otros programadores han escrito SW a nivel de usuario
  - Gratis en Internet o bien comprar alguna de las distribuciones
  - No tenía competencia: al mismo tiempo que nacía, salía FreeBSD pero por problemas judiciales no pudo despegar

# 7.1. Historia de UNIX/Linux (vi)

## Curiosidad...

<http://www.oreilly.com/catalog/opensources/book/appa.html>

**From:** ast@cs.vu.nl (Andy Tanenbaum)

**Newsgroups:** comp.os.minix

**Subject:** LINUX is obsolete

**Date:** 29 Jan 92 12:12:50 GMT

... As most of you know, for me MINIX is a hobby... As a result of my occupation, I think I know a bit about where operating are going in the next decade or so... monolithic vs. microkernel...

**From:** torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)

**Subject:** Re: LINUX is obsolete

**Date:** 29 Jan 92 23:14:26 GMT

your job is being a professor and researcher: That's one hell of a good excuse for some of the brain-damages of minix. I can only hope (and assume) that Amoeba doesn't suck like minix does.

## 7.2. Estructura de UNIX/Linux

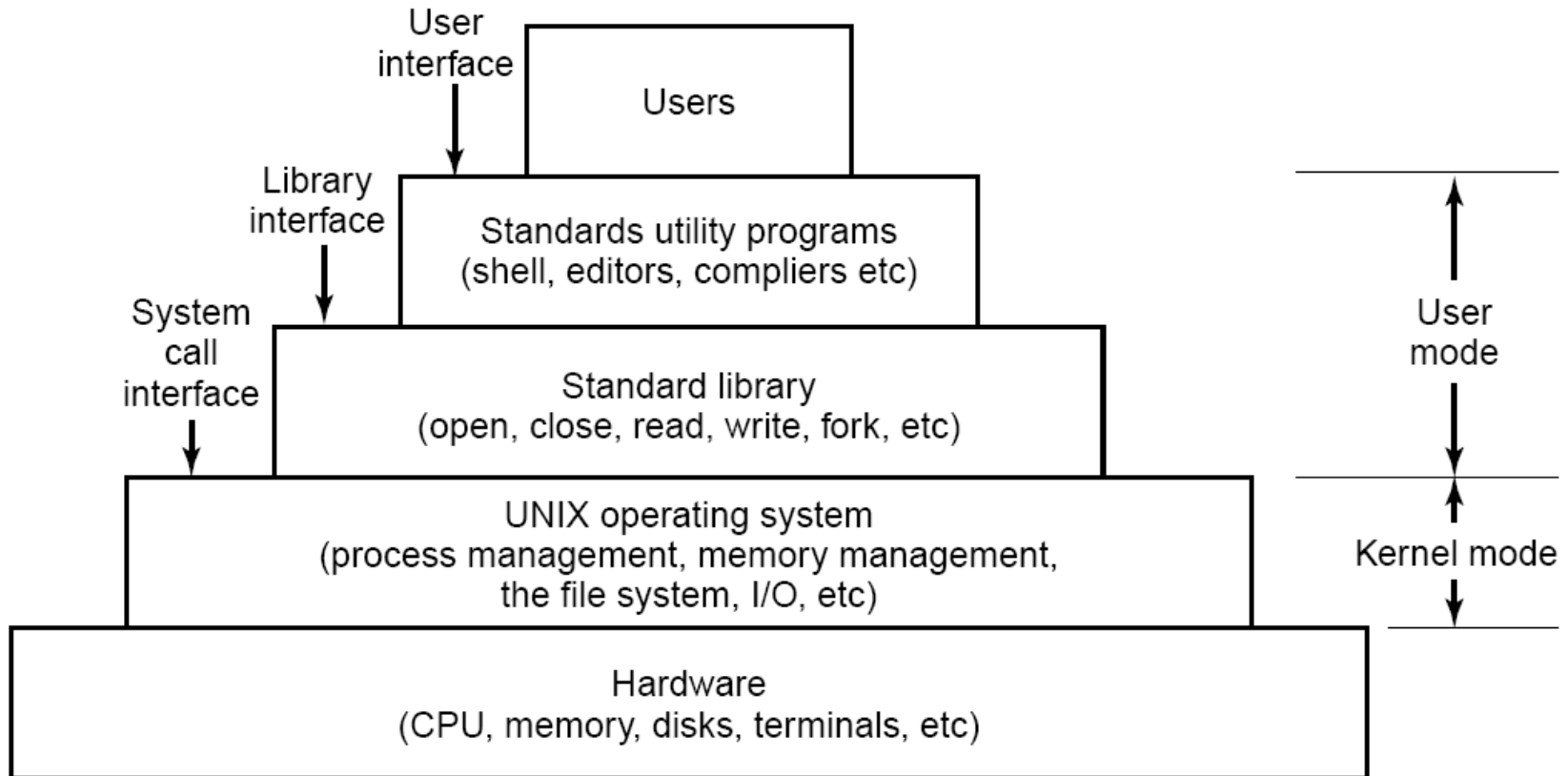
### 7.2.1 Metas de diseño

- Diseñado para manejar múltiples procesos y múltiples usuarios
- Sencillo, elegante y consistente
- *Principio de la mínima sorpresa*
- Potencia y flexibilidad ) número reducido de elementos básicos, que se pueden combinar de muchas formas
- Un programa debe hacer una sola cosa y hacerla bien
- Evitar redundancia innecesaria: mejor *cp* que *copy*
- Sistema diseñado por programadores para programadores

# 7.2. Estructura de UNIX/Linux (i)

## 7.2.2. Interfaz de UNIX/Linux

- Se ejecuta sobre hardware desnudo ) controlar el HW y proporcionar a todos los programas una interfaz de llamadas al sistema



## 7.2. Estructura de UNIX/Linux (i)

### 7.2.3. Características de Linux

- Completamente de 32 bits (existen versiones de 64 bits)
- Multitarea y multiusuario. Y tiene protección entre procesos
- Multiplataforma, (Intel, SPARC, Alpha, PowerPC, etc.)
- Carga de ejecutables por demanda
- Memoria virtual usando paginación
- Librerías compartidas de carga dinámica (DLL's) y librerías estáticas
- Consolas virtuales múltiples
- Sistema de ficheros *ext2*
- Soporte para redes TCP/IP con IPv4 e IPv6. Puede actuar como router, firewall, proxy, etc.
- Soporte para protocolos de red como AX25, X.25, IPX/SPX, SMB (o NetBEUI), DDP, SLIP/PPP, PLIP, etc.
- Ejecución de binarios ELF, y otros formatos de UNIX
- etc.

## 7.2. Estructura de UNIX/Linux (ii)

- 7.2.4. Kernel de Linux

| Llamadas al sistema                       |                      |                                    |                                       |                  |                          |                                    |
|---|----------------------|------------------------------------|---------------------------------------|------------------|--------------------------|------------------------------------|
| Manejo de terminales                      |                      | Sockets                            | VFS                                   | Corres-pondencia | Manejo de señales        | Creación y eliminación de procesos |
| Tty<br>cruda                              | Tty cocinada         | Protocolos de red                  | Sistemas de ficheros                  | Memo. virtual    |                          |                                    |
|   | Disciplinas de línea | Enrutamiento                       | Caché de buffer                       | Caché de páginas | Planificador de procesos |                                    |
| Manejadores de dispositivos de caracteres |                      | Manejadores de dispositivos de red | Manejadores de dispositivos de discos |                  | Despachador de procesos  |                                    |
| Hardware                                  |                      |                                    |                                       |                  |                          |                                    |

## 7.2.4. Kernel de Linux

- La capa inferior está formada por los manejadores de dispositivos y el despachador de procesos:
  - **Manejadores para dispositivos de caracteres**
  - **Manejadores para dispositivos de bloques**, en los que se puede realizar búsquedas por bloques
  - **Manejadores para dispositivos de red**, que son dispositivos de caracteres pero se manejan de forma distinta
  - El **despachador** que se ejecuta cuando se presenta una interrupción y se encarga de realizar los cambios de contexto, deteniendo al proceso en ejecución, guardando su estado en la tabla de procesos, e iniciando el manejador apropiado o a otro proceso de usuario

## 7.2.4. Kernel de Linux (i)

- La siguiente capa está dividida en 4 columnas:
  - **Dispositivos de caracteres**, con dos modos de funcionamiento:
    - **Modo crudo**: orientado a caracteres, los programas reciben cada pulsación tan pronto como se pulsa la tecla sin ser tratada, (un ejemplo son: *vi*, *emacs*)
    - **Modo cocinado**: orientado a caracteres, los programas reciben la línea completa, que ha sido editada y corregida mediante la disciplina de líneas

## 7.2.4. Kernel de Linux (ii)

- Continúa. . .
  - **Software de red:** es modular, reconoce diferentes dispositivos, y está formado por tres subcapas:
    - Capa de **enrutamiento**, enruta los paquetes para que lleguen al dispositivo o manejador de protocolo correcto
    - Capa de **protocolos de red**, encima de la anterior, incluye los protocolos *IP* y *TCP*, y además puede incluir otros protocolos
    - **Interfaz de sockets**, superpuesta a toda la red, permite a los programas crear sockets para redes y protocolos específicos

## 7.2.4. Kernel de Linux (iii)

- Continúa. . .
  - Encima de los manejadores de discos:
    - **Caché de páginas** y la **caché de buffer** dentro de la misma
    - Los **sistemas de archivos** están encima del caché de *buffers*, y maneja los múltiples sistemas de ficheros que acepta Linux
    - Encima de los sistemas de archivos está el VFS, (*virtual file system*), encargado de la asignación de nombres, la administración de directorios, enlaces simbólicos, y otras propiedades que son iguales en todos los sistemas de ficheros
    - El **sistema de memoria virtual** está encima de la caché de páginas, y es donde se encuentra toda la lógica de paginación, como el reemplazo de páginas
    - Dentro de la memoria virtual se manejan los **fallos de página**
    - Por último, **la correspondencia** de ficheros en la memoria virtual

## 7.2.4. Kernel de Linux (iv)

- Continúa. . .
  - Encima del despachador están:
    - El **planificador de procesos**, que elige el siguiente proceso a ejecutar. Si los hilos se administran a nivel del Kernel, su administración también se efectúa aquí
    - Código para procesar **señales** y enviarlas al destino correcto
    - Código para **crear y terminar procesos**
- La capa superior es la interfaz con el sistema: las **llamadas al sistema**

# 8. Introducción a Windows 2000

## Historia de Windows 2000

- MS-DOS ⇒ Windows 95/98/Me
- Windows NT (*Windows Nueva Tecnología*) ⇒ pensado para aplicaciones de negocios y usuarios “caseros”
- Windows 2000 ⇒ Windows NT 5.0 en un intento de tener un único sistema operativo (*Basado en VMS*)
- Sistema multiprogramado de 32 y 64 bits con procesos protegidos individualmente
- S.O. en modo kernel, procesos de usuario en modo usuario
- Soporte para ejecutarse en multiprocesadores simétricos de hasta 32 procesadores

## 8.1. Historia de Windows 2000 (i)

- Soporte para dispositivos Plug and Play, bus USB, IEEE 1394 (FireWire), IrDA, administración de energía, etc.
- Active Directory, seguridad con Kerberos, manejo de tarjetas inteligentes, etc.
- El sistema de ficheros NTFS maneja archivos cifrados, comprimidos, cuotas, archivos enlazados, etc.
- Internalización: un único binario que permite elegir idioma
- Trabaja con Unicode, (poder manejar idiomas como ruso, hebreo, etc.)
- No usa MS-DOS como plataforma base.
- Sólo para dos plataformas Pentium e Intel IA-64

# 8.1. Historia de Windows 2000 (ii)

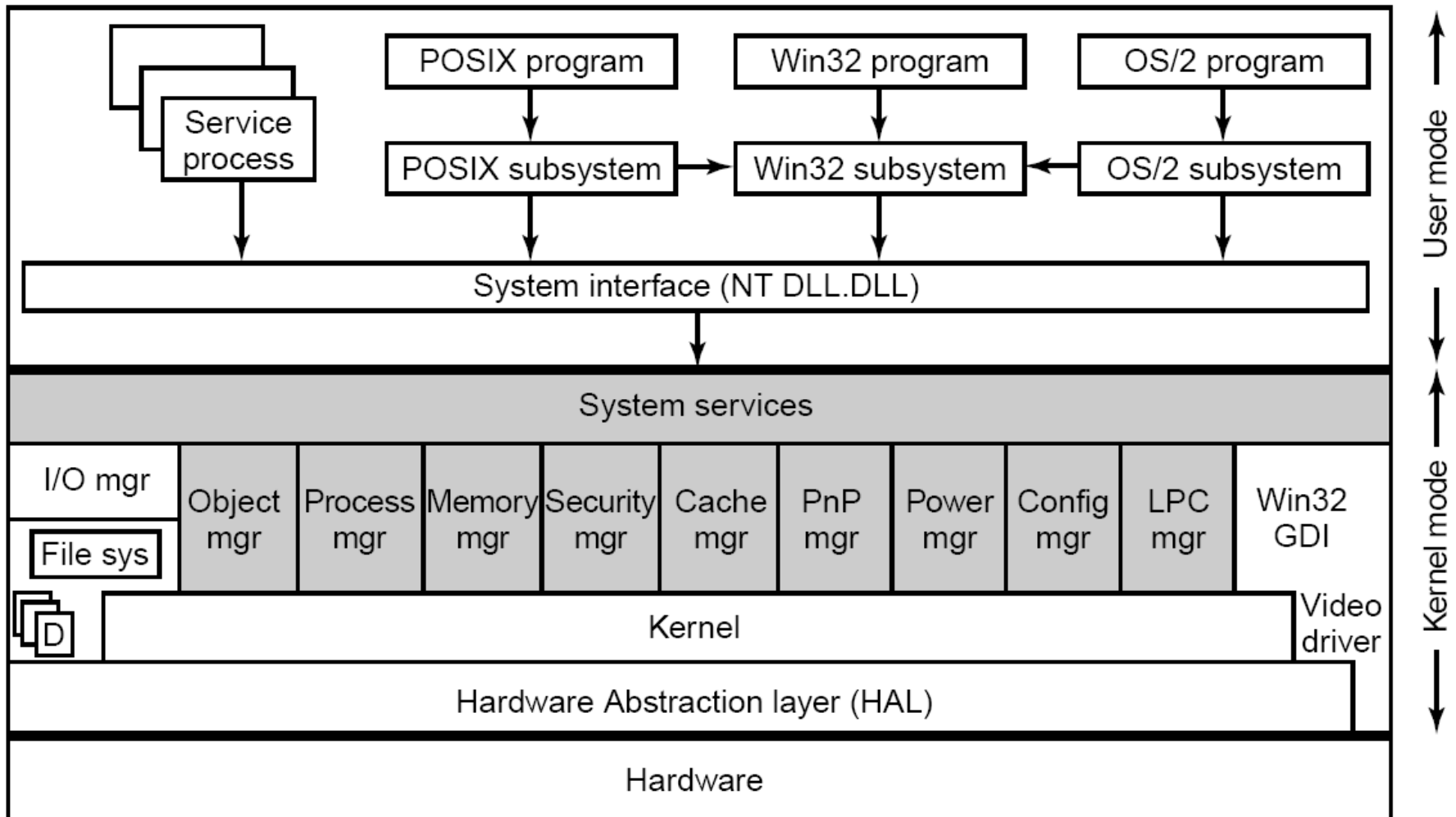
| Version           | Max RAM | CPUs | Max clients | Cluster size | Optimized for |
|-------------------|---------|------|-------------|--------------|---------------|
| Professional      | 4 GB    | 2    | 10          | 0            | Response time |
| Server            | 4 GB    | 4    | Unlimited   | 0            | Throughput    |
| Advanced server   | 8 GB    | 8    | Unlimited   | 2            | Throughput    |
| Datacenter server | 64 GB   | 32   | Unlimited   | 4            | Throughput    |

- **Todas las versiones usan el mismo binario**
- **Las variables *ProductType* y *ProductSuite* indican cómo ha de funcionar ese binario**
- **Microsoft ha desarrollado varios kits de herramientas para usuarios avanzados:**
  - Herramientas de apoyo
  - **SDK, *Kit de desarrollo Software***
  - **DDK, *Kit para desarrollo de controladores de dispositivos***
  - **Kit de Recursos**

## 8.2. Estructura de Windows 2000

- Dividido en dos partes:
  - S.O. se ejecuta en modo núcleo
  - Subsistemas de entorno, (en modo usuario), son procesos individuales que ayudan a los procesos de usuario a realizar ciertas funciones del sistema
- Diseño por capas, algunas capas divididas en módulos
- Cada módulo tiene una función específica y una interfaz bien definida
- Las capas HAL y Kernel escritas en C y ensamblador
- Las otras capas están escritas en C, y son casi independientes del hardware
- Los controladores escritos en C, y algunos en C++

## 8.2. Estructura de Windows 2000 (i)



## 8.2.1. Capa de abstracción hardware, HAL

- Capa clave  $\Rightarrow$  W2K portable entre distintas plataformas
- Presenta dispositivos de HW abstractos y privados (oculta los detalles de múltiples arquitecturas)
- Dispositivos  $\Rightarrow$  servicios independientes de la máquina, que usan el resto del S.O. y los manejadores de dispositivos
- Los servicios de HAL incluyen:
  - Acceso a los registros del dispositivo
  - Direccionamiento de dispositivos independiente del bus
  - Manejo y restablecimiento de interrupciones
  - Transferencia por DMA
  - Control de los temporizadores y del reloj de tiempo real
  - Candados de exclusión mútua y sincronización de multiprocesadores
  - Comunicación con la BIOS y la CMOS
  - No proporciona servicios para dispositivos de E/S específicos como teclados, ratones, discos o memoria

## 8.2.2. Capa del Kernel

- Construye una abstracción de más alto nivel, haciendo que el resto del S.O. sea independiente del hardware y muy portable
- Consiste en un conjunto de funciones que proporcionan mecanismos fundamentales como:
  - Realización de cambios de contexto
  - Planificación de procesos
  - Manejador de interrupciones
  - Proporciona apoyo de bajo nivel para dos clases de objetos internos del S.O.:
    - Objetos de *control*: controlan varios aspectos del S.O., incluyen objetos de manejadores de E/S, el objeto de interrupción, y los objetos APC y DPC (estos ejecutan procesos para los que el tiempo no es crítico)
    - Objetos *despachador*: incorporan capacidades de sincronización (semáforos, mutex, etc.) que pueden alterar o afectar a la planificación de procesos

## 8.2.3 El ejecutivo

- Está escrito en C, es independiente de la arquitectura y es portable “sin mucho esfuerzo”
- Formado por 10 componentes, cada uno de los cuales es una colección de procedimientos que colaboran para alcanzar una meta
- No hay ocultación de la información
- Los componentes que lo forman son:
  - **Administrador de objetos:** crea, maneja y elimina todos los objetos que reconoce el S.O., (procesos, procesos hilos, ficheros, dispositivos de E/S, etc.)
  - **Administrador de E/S:** administra dispositivos de E/S y presta servicios genéricos de E/S, haciendo que sea independiente del dispositivo

## 8.2.3. El ejecutivo (i)

- Componentes del ejecutivo, continúa. . .
  - **Administrador de procesos:** maneja los procesos y los hilos, incluidas su creación y terminación, y se ocupa de los mecanismos para administrarlos
  - **Administrador de memoria:** implementa la arquitectura de memoria virtual paginada por demanda
  - **Administrador de seguridad:** hace cumplir los mecanismos de seguridad
  - **Administrador de caché:** administra la memoria caché de disco, manteniendo en memoria los bloques de disco más recientemente utilizados
  - **Administrador de Plug-and-Play:** determina qué manejador es necesario para manejar un dispositivo y lo carga. Recibe todas las notificaciones de dispositivos nuevos conectados

## 8.2.3. El ejecutivo (ii)

- Componentes del ejecutivo, continúa. . .
  - **Administrador de consumo de electricidad:** trata de reducir al mínimo el consumo de energía
  - **Administrador de configuración:** es el encargado del *Registro*, añade nuevas entradas y consulta claves
  - **Administrador de llamadas a procedimientos locales:** ofrece un mecanismo de comunicación entre procesos, usado entre los procesos y sus subsistemas
  - **Módulo GDI Win32:** maneja ciertas llamadas al sistema, administra imágenes para el monitor y las impresoras, y contiene el administrador de ventanas y el manejador de la pantalla
  - **Servicios del sistema:** su función es proporcionar una interfaz con el ejecutivo, acepta las verdaderas llamadas al sistema de W2K e invoca a partes del ejecutivo para que las ejecute

## 8.2.4. Manejadores de dispositivos

- Cada uno puede controlar uno o más dispositivos de E/S
- También pueden hacer tareas no relacionadas, como cifrar un flujo de datos o proporcionar acceso a estructuras de datos del kernel
- No forman parte del binario del kernel (este es el mismo para todo el mundo)
- Al instalarse se añaden a una lista del *Registro* y se cargan dinámicamente cuando arranca el sistema
- Los sistemas de ficheros también están presentes como manejadores de dispositivos
- Un manejador de dispositivo mal diseñado puede afectar al kernel, y corromper el sistema

## 8.2.5. Subsistema de entorno

- Está formado por tres componentes: DLLs, subsistemas de entorno, y procesos de servicio
- Se ejecutan en **modo usuario**
- Las **DLLs** y los **subsistemas de entorno** implementan la funcionalidad de la interfaz publicada (distinta de la de llamadas al sistema)
- Dentro de los subsistemas de entorno hay tres distintos:
  - **Win32** (oficial de W2K)
  - **POSIX**: ofrece soporte mínimo para aplicaciones UNIX
  - **OS/2**: ofrece soporte mínimo para aplicaciones OS/2
- Los **procesos de servicio** organizan servicios tales como impresión, planificación de tareas, etc.