

# Fundamentos de los Sistemas Operativos

## Tema 1. Conceptos generales Estructura del sistema operativo

© 2015 ULPGC - José Miguel Santos Espino

# Contenido

- Componentes del S.O.
  - Programas del sistema
  - El núcleo
  - Llamadas al sistema
- Arquitecturas para los S.O.
  - Ejemplos: sistemas monolíticos, por capas
  - Micronúcleos
  - Módulos cargables

# Subsistemas típicos de un SO

- Procesos e hilos
- Entrada/salida
- Memoria y cachés de disco
- Archivos
- Red y mensajería
- Usuarios y seguridad
- Monitorización y contabilidad

# Programas del sistema

- Un SO es una plataforma de software que suele incluir un conjunto de utilidades básicas, para:
  - Darnos un entorno de trabajo (escritorio, shell...)
  - Gestionar los recursos (formatear discos, configurar la red...)
  - Trabajar con archivos (ls, cp, mkdir...)
  - Editar documentos (vi, notepad, gedit...)
  - Desarrollar programas (compilador, depurador...)
- Son lo que los usuarios perciben como «sistema operativo»

# Programas del sistema

- Otros programas del sistema son servicios que se ejecutan en segundo plano (*servicios, subsistemas, demonios*):
  - Sistema de impresión
  - Copias de seguridad
  - Registro de actividad
  - ...

# El núcleo (*kernel*)

- Se suele llamar **núcleo** al componente del SO que reside en memoria de forma permanente y atiende las llamadas al sistema y demás eventos.
- El resto de utilidades del SO (CLI, GUI, programas del sistema...) se apoyan en los servicios del núcleo.
- En la parte teórica de la asignatura trataremos casi exclusivamente sobre el núcleo, no abordaremos los otros componentes.

# Implementación de las llamadas al sistema

- En el nivel del procesador:
  - la llamada al sistema ocurre mediante una instrucción especial del procesador (syscall, int, trap...)
  - esa instrucción cambia a modo privilegiado
- En el ámbito del programador:
  - La llamada es una subrutina que escribimos en el código fuente. El compilador la acabará sustituyendo por una invocación a la instrucción especial, con los argumentos que sean necesarios.

# Ejemplos de llamadas al sistema

- Windows:

```
handle = OpenFile("mifichero",ofstruct,OF_READ);
```

- UNIX:

```
fd = open("mifichero",O_RDONLY);
```

- MS-DOS:

```
mov ah,3Dh
```

```
mov al,0
```

```
mov dx,StringMiFichero
```

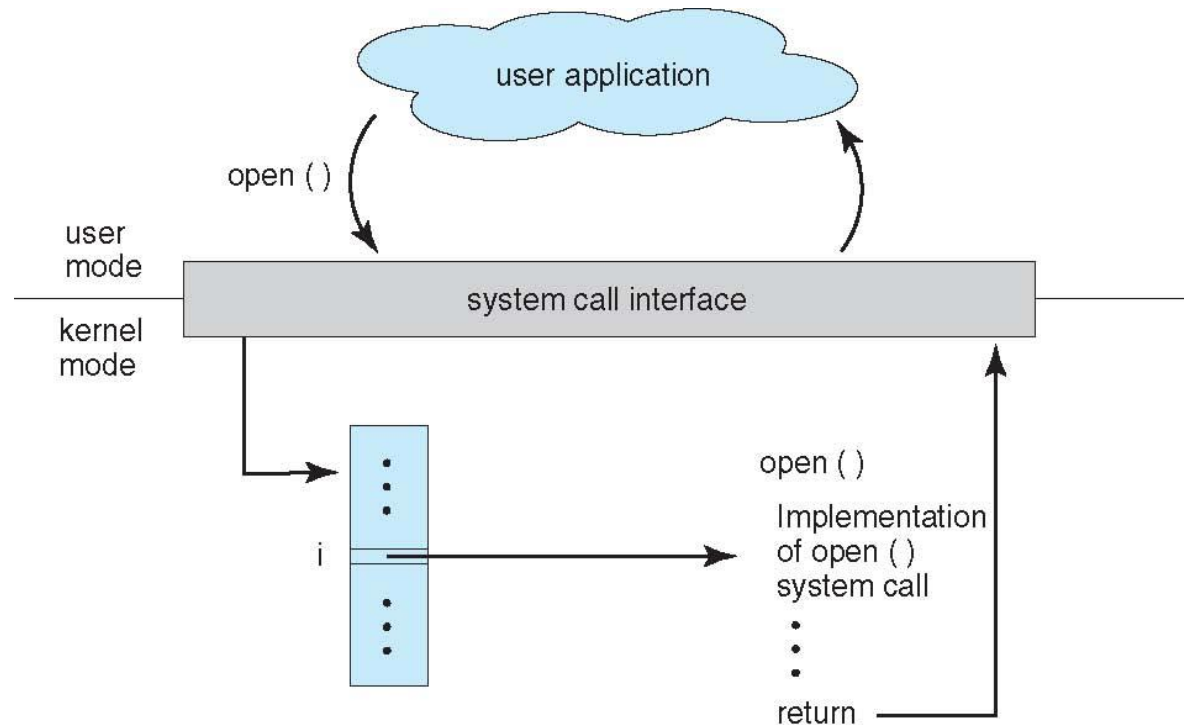
```
int 21h
```



# Implementación de las llamadas al sistema

- ¿Cómo se pasan los argumentos a la llamada?
  - Mediante registros de CPU (lo más típico)
  - Escribiéndolos en una tabla en memoria principal
  - Colocándolos en la pila

# Relación entre el proceso, la API de llamadas al sistema y el núcleo



*(tomado de Silberschatz, Galvin & Gagne, 2013)*

# Ejemplos de llamadas al sistema (UNIX)

- Procesos: crear proceso (fork), finalizar proceso (exit)...
- Memoria: pedir más memoria (sbrk), liberar memoria...
- Archivos: open, close, creat, read, write, mkdir; bloquear fichero (lockf)...
- Redes: crear conexión (socket), cerrar conexión...
- Protección de ficheros: cambiar permisos (chmod), cambiar propietario (chown)...

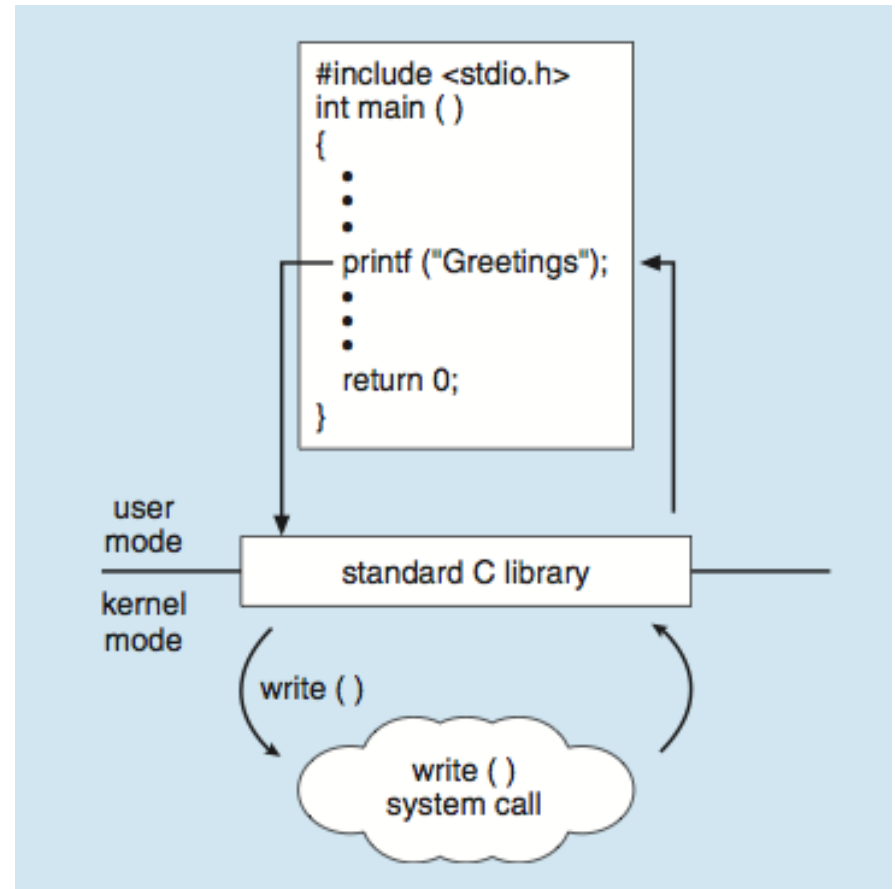
# Ejemplos en Windows y UNIX

	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 & Gagne, 2013)*

# Ejemplo con la biblioteca estándar

- La biblioteca estándar de C no pertenece al SO y se ejecuta en modo usuario.
- P.ej. **printf()** utiliza la llamada al sistema **write()** para poder escribir en la consola.



# Arranque típico de un SO

1. Cuando el equipo se enciende, la CPU inicia su ejecución en un punto fijo de la memoria
2. Hay una ROM con una pequeña rutina de arranque
3. La rutina localiza en qué dispositivo se encuentra el **cargador** del SO (*boot loader*) y lo carga en memoria
  - Nota: La ROM del equipo tiene código para leer y escribir sobre los dispositivos de E/S
4. El cargador instala el **núcleo** y se continúa el proceso de carga de módulos, servicios, etc. hasta que el SO queda totalmente operativo.

# ARQUITECTURAS PARA LOS S.O.

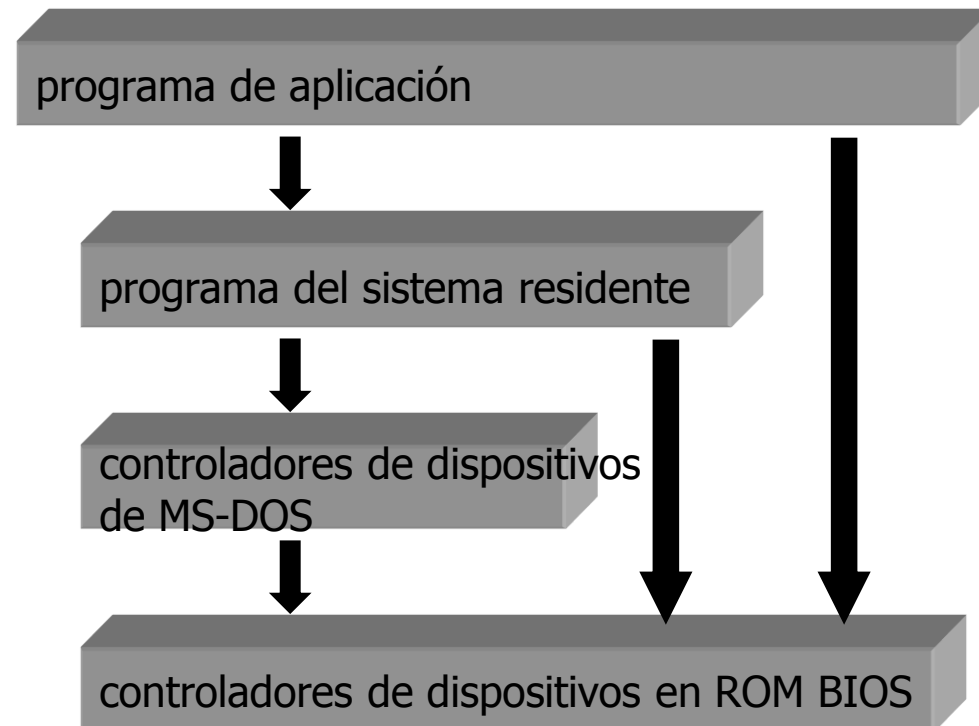
# Arquitectura del SO

- ¿qué estructura interna tiene un SO?
- Algunas estructuras:
  - Monolítico → todo en un mazacote de código
  - En capas → niveles de abstracción creciente
  - Micronúcleos → un pequeño SO sobre el que se añaden módulos



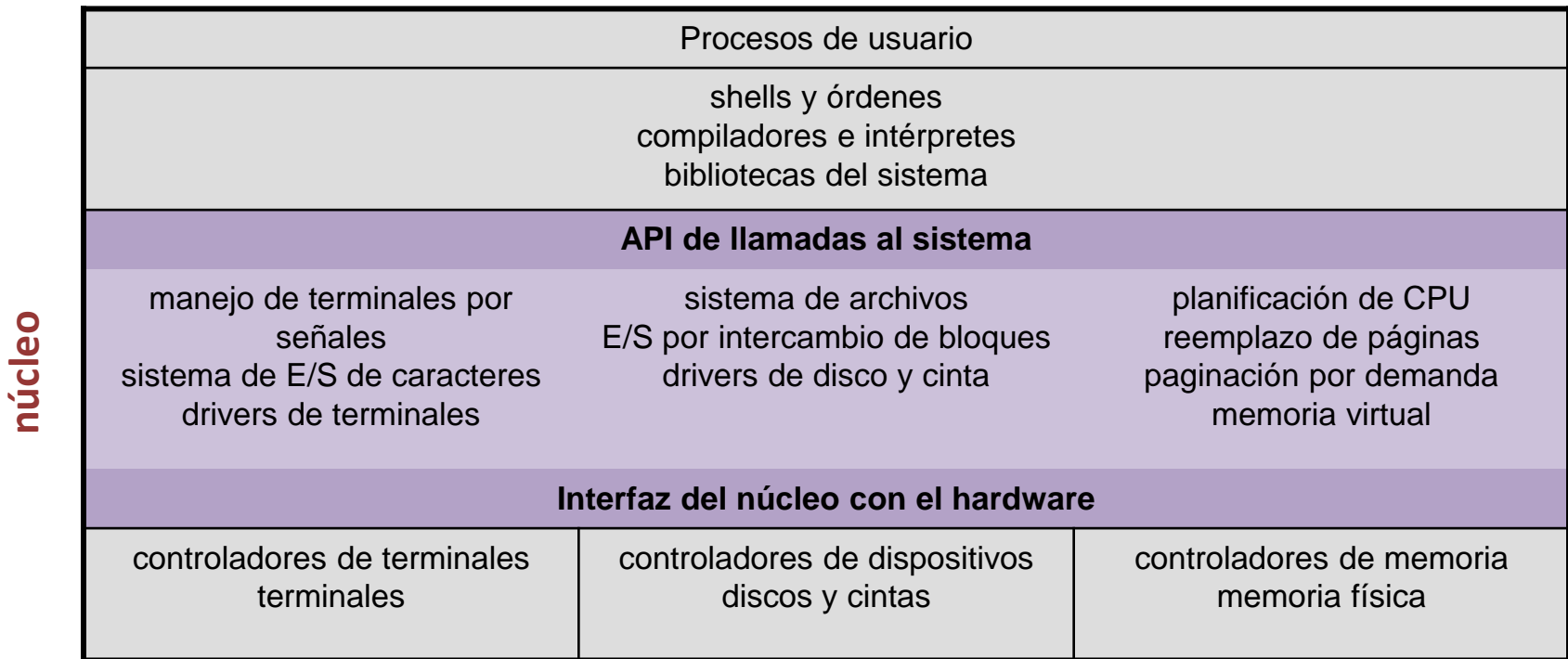
# Sistema monolítico sencillo (MS-DOS)

- Estructura ligeramente modular



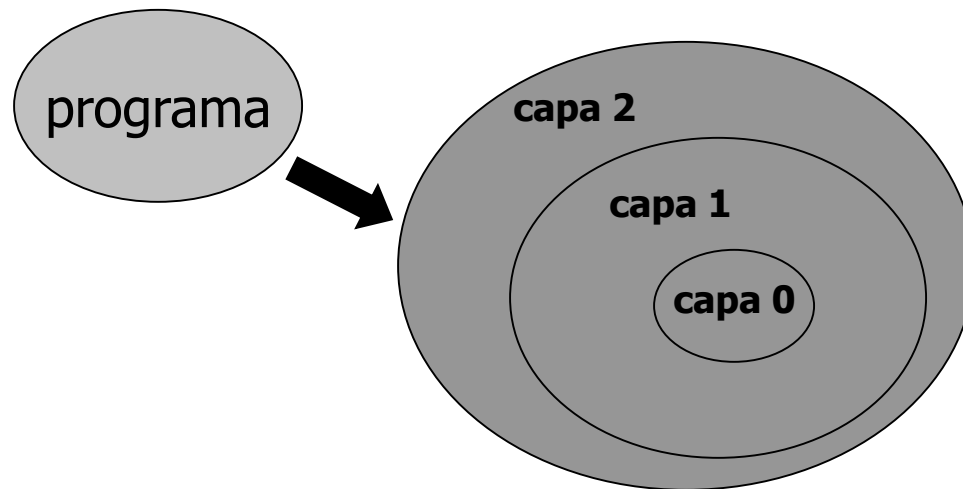
# Sistema monolítico complejo (UNIX clásico)

- Separación en capas más nítida



# Diseño por capas

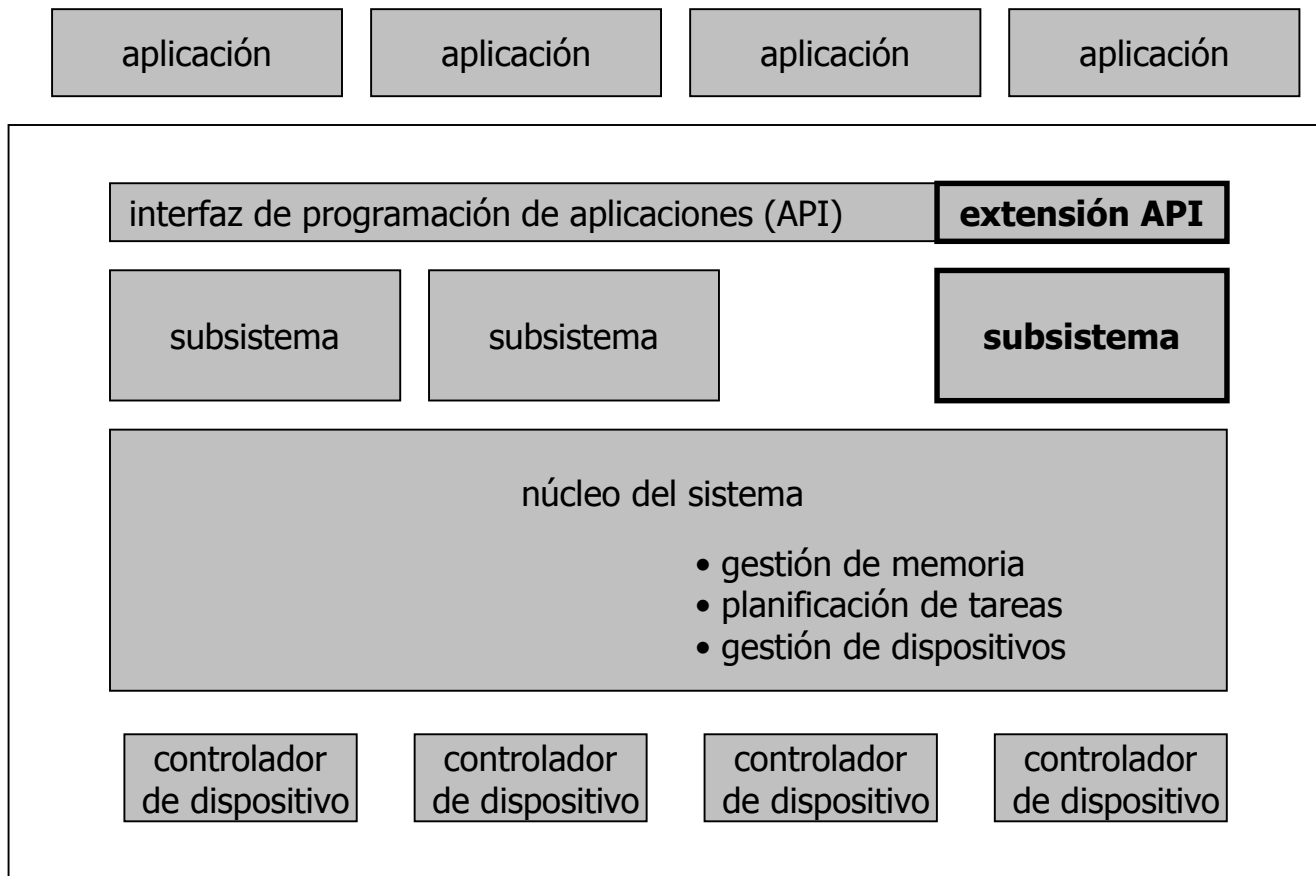
- El SO se construye como una jerarquía de niveles, cada uno de los cuales aprovecha los servicios del nivel inferior.



# Sistema por capas puro (THE)

- Sistema experimental de los años 60
- Seis niveles:
  - L5: aplicaciones de usuario
  - L4: buffering
  - L3: consola del operador
  - L2: gestión de memoria paginada
  - L1: planificación de procesos
  - L0: hardware

# Sistema por capas (Windows, OS/2)



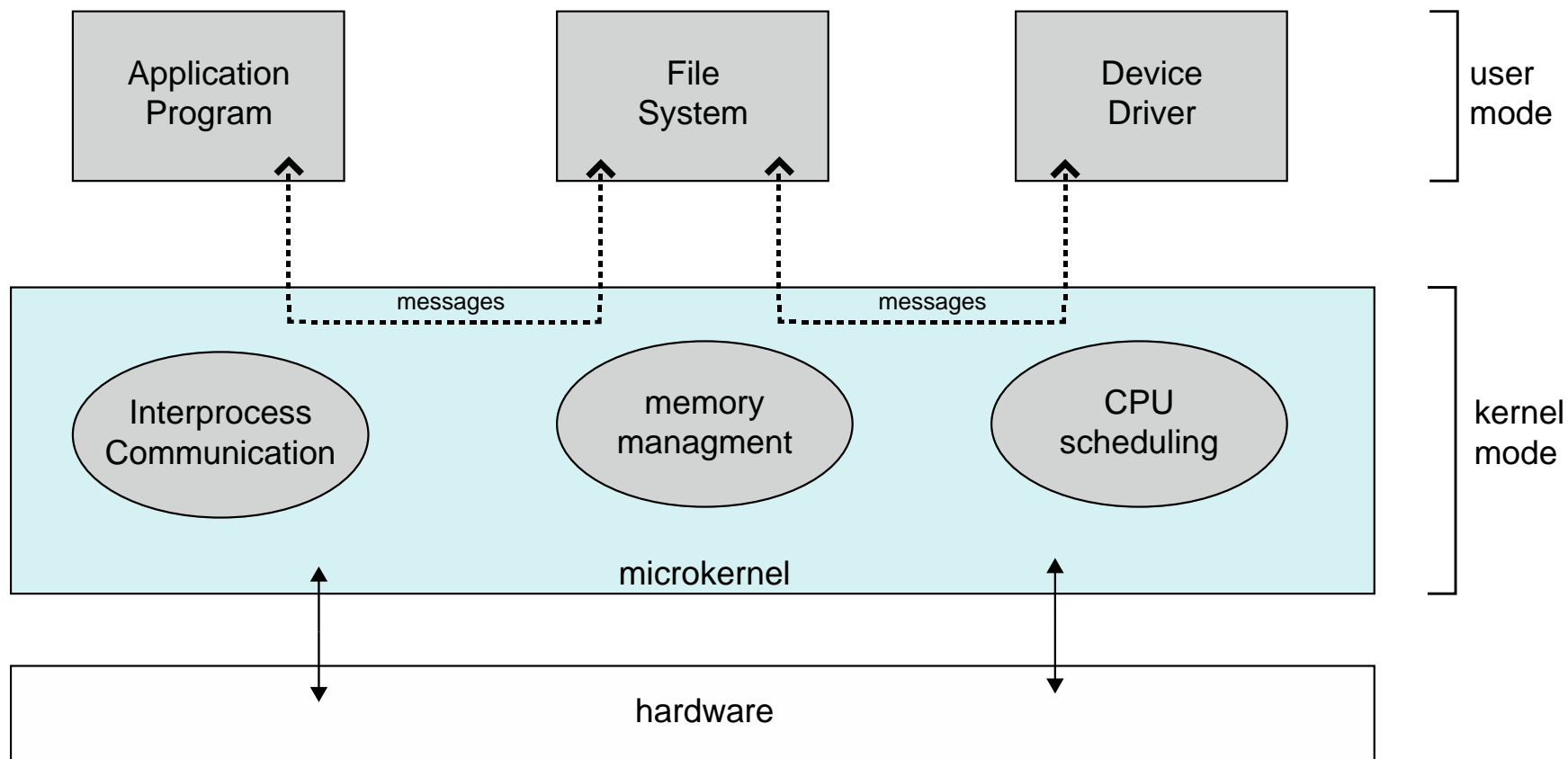
# Separar mecanismos y políticas

- Políticas → estrategias para administrar un recurso
  - SJF, FIFO, tiempo compartido, LRU...
- Mecanismos → estructuras y objetos que sirven para implementar una política
  - Colas de espera, bitmaps, despachador de procesos, temporizador, etc.
- Deseable que el código de los mecanismos y el de las políticas estén separados. Así se pueden reutilizar los mecanismos para otras políticas.

# Micronúcleos

- Dejar en el núcleo lo mínimo imprescindible
  - Multitarea básica, gestión de interrupciones, comunicación entre procesos, E/S, etc.
- El resto de servicios se implementan como módulos en espacio de usuario
- Mecanismos en el micronúcleo, políticas fuera
- Los módulos se comunican con mensajes
- Primer micronúcleo: **Mach** (1980)

# Micronúcleo: ejemplo Mach





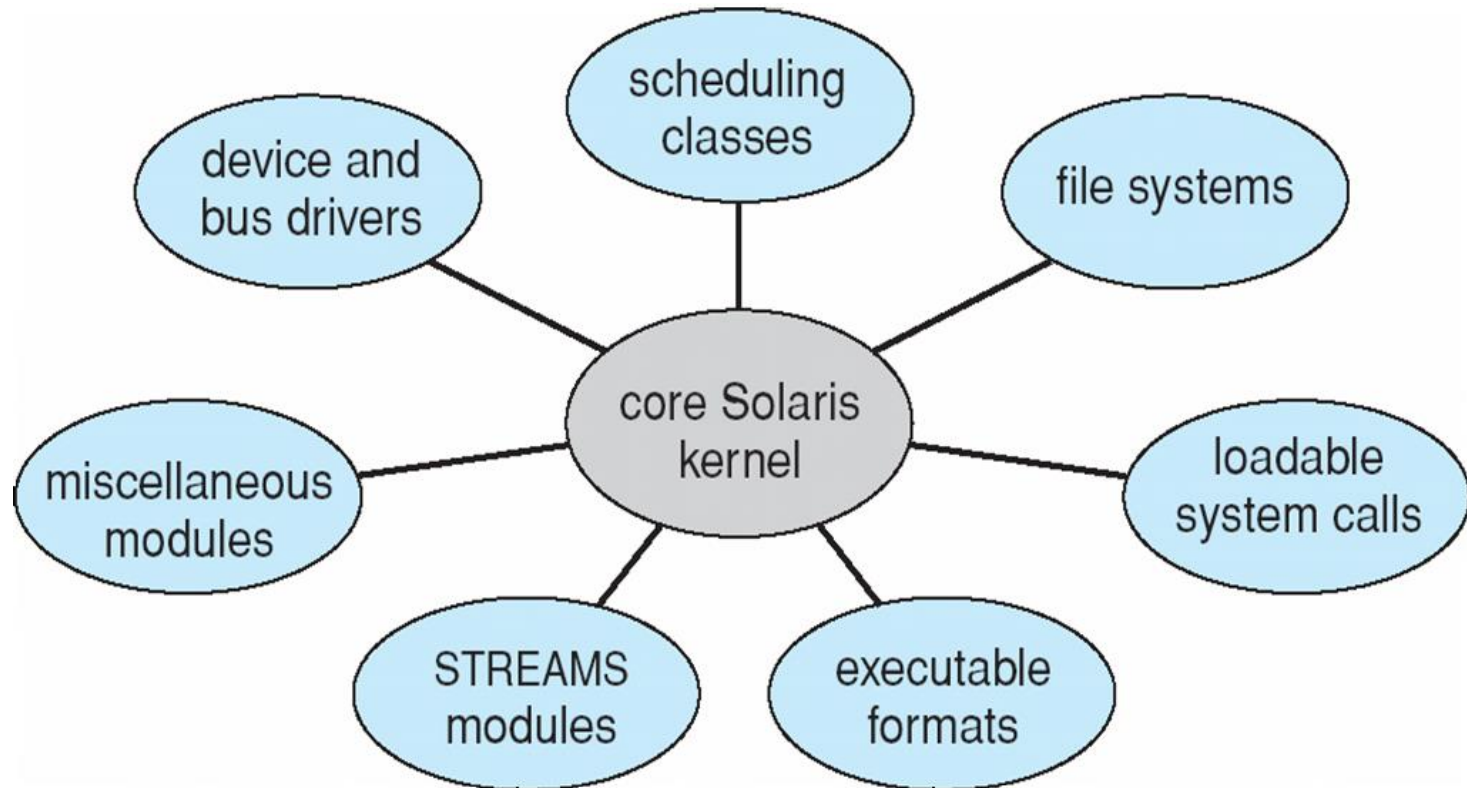
# Micronúcleos: ventajas

- Ventajas
  - Se pueden construir servicios nuevos del SO sin tocar el núcleo
  - Se pueden implementar múltiples versiones para un mismo servicio (ej. varios sistemas de ficheros)
  - El SO es más fácil de portar a otras arquitecturas (sólo hay que tocar el micronúcleo)
  - Más seguridad y fiabilidad (menos código en modo privilegiado)
- Inconvenientes
  - La comunicación entre módulos penaliza el rendimiento

# Módulos cargables

- Linux, Solaris...
- Módulos de código que se pueden incorporar al núcleo en tiempo de ejecución
- Cada módulo tiene una interfaz conocida
- Los módulos se hacen llamadas entre ellos

# Módulos cargables (Solaris)



# Implementación del SO

- El SO presenta características especiales
  - Pieza crítica: todas las aplicaciones dependen de él
  - Es mucho más complicado de depurar y de actualizar
- Lenguaje de programación
  - En el pasado, en lenguaje ensamblador (por eficiencia)
  - Ahora en lenguajes de alto nivel, sobre todo C y C++
- Desarrollo modular
  - Posibilidad de desarrollar módulos cargables por separado y dinámicamente
- Disponibilidad del código fuente
  - Sistema propietario → sólo tenemos los binarios ya compilados (Windows, Mac OS X, OS/360...)
  - Software libre → podemos tener el fuente y compilarlo en nuestro equipo (Linux, FreeBSD...)

# FIN del Tema 1

© 2015 ULPGC – José Miguel Santos Espino