Introducción

Módulo 1
2016-2017

¿Necesitamos un sistema operativo?

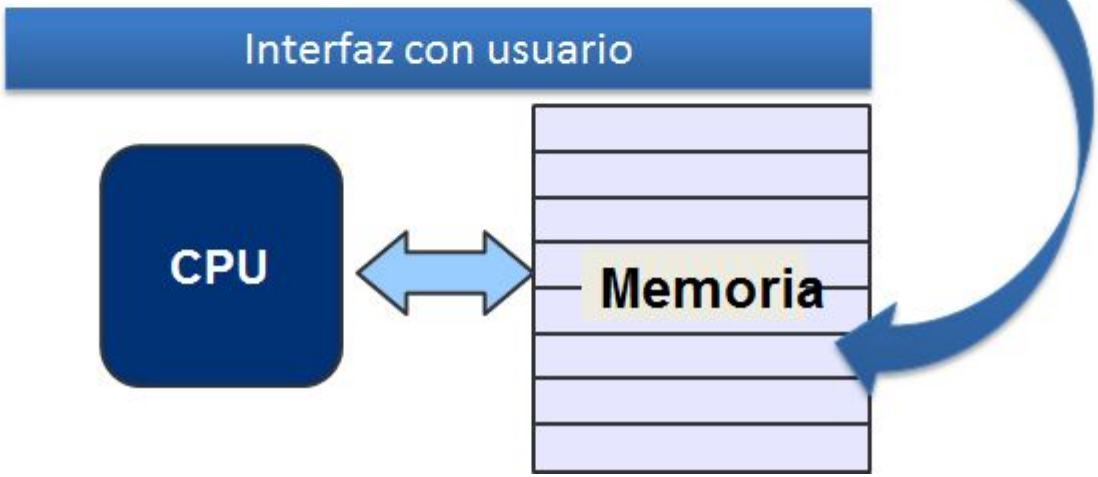
- A estas alturas ya sabemos….
  - Programar en lenguajes de alto nivel
  - Programar en ensamblador
  - Gestionar la E/S a bajo nivel
- ¿Qué ha hecho el sistema operativo por nosotros todo este tiempo?
  - Veámoslo como un proveedor de servicios

Repasemos lo que ya sabemos ...

```
program simple;
var
   a, b, p:
integer
begin
   a := 5;
   b := 3;
   p := a * b;
end
```

```
simple:
  leal   4(%esp), %ecx
  andl   $-16, %esp
  pushl  -4(%ecx)
  pushl  %ebp
  movl   %esp, %ebp
  pushl  %ecx
  subl   $16, %esp
  movl   $5, -16(%ebp)
  movl   $3, -
12(%ebp).....
```

```
457f 464c 0101 0001 0000 0000 0000 0000
0002 0003 0001 0000 8280 0804 0034 0000
0df0 0000 0000 0000 0034 0020 0007 0028
0022 001f 0006 0000 0034 0000 8034 0804
8034 0804 00e0 0000 00e0 0000 0005 0000
0004 0000 0003 0000 0114 0000 8114 0804
8114 0804 0013 0000 0013 0000 0004 0000
0001 0000 0001 0000 0000 0000 8000 0804
8000 0804 046c 0000 046c 0000 0005 0000
1000 0000 0001 0000 046c 0000 946c 0804
946c 0804 0100 0000 0104 0000 0006 0000
1000 0000 0002 0000 0480 0000 9480 0804
9480 0804 00c8 0000 00c8 0000 0006 0000
0004 0000 0004 0000 0128 0000 8128 0804
8128 0804 0020 0000 0020 0000 0004 0000
```



¿Cómo se ejecuta mi código?

i) El usuario hace doble click sobre el icono del programa que quiere ejecutar

DATOS PROGRAMA

ii) El sistema operativo busca el programa en las unidades de almacenamiento

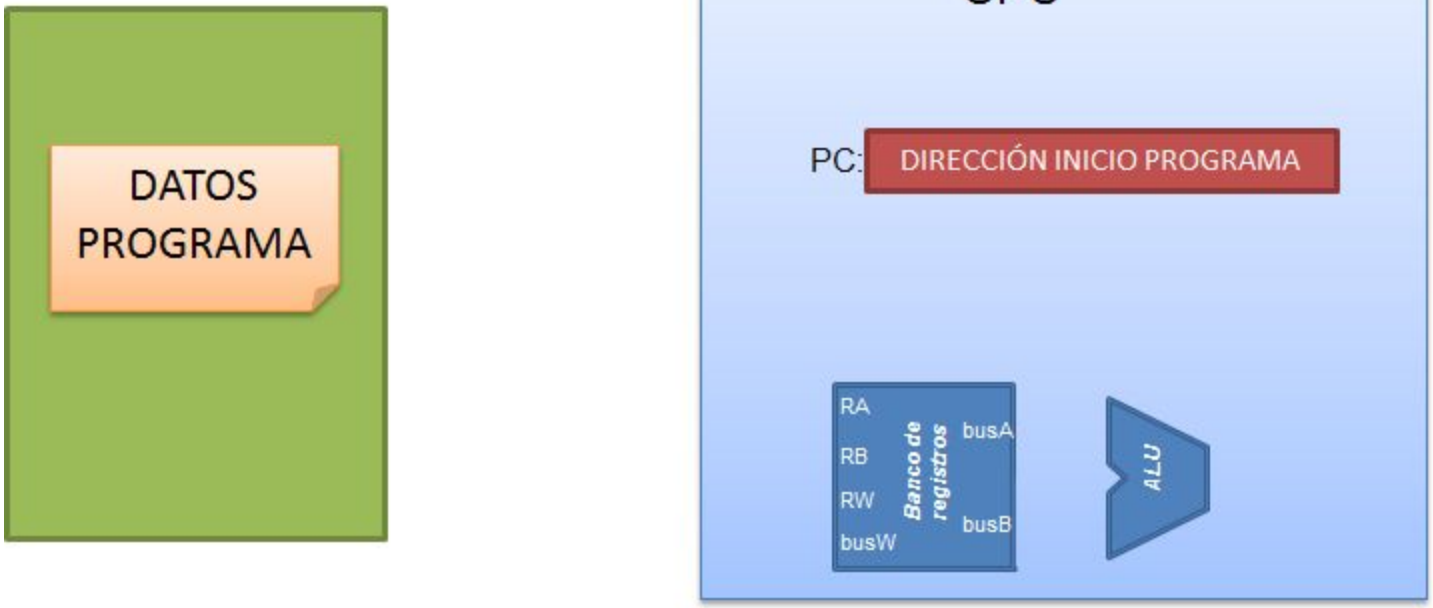
RAM

DIRECCIÓN DE INICIO DEL PROGRAMA

DATOS PROGRAMA

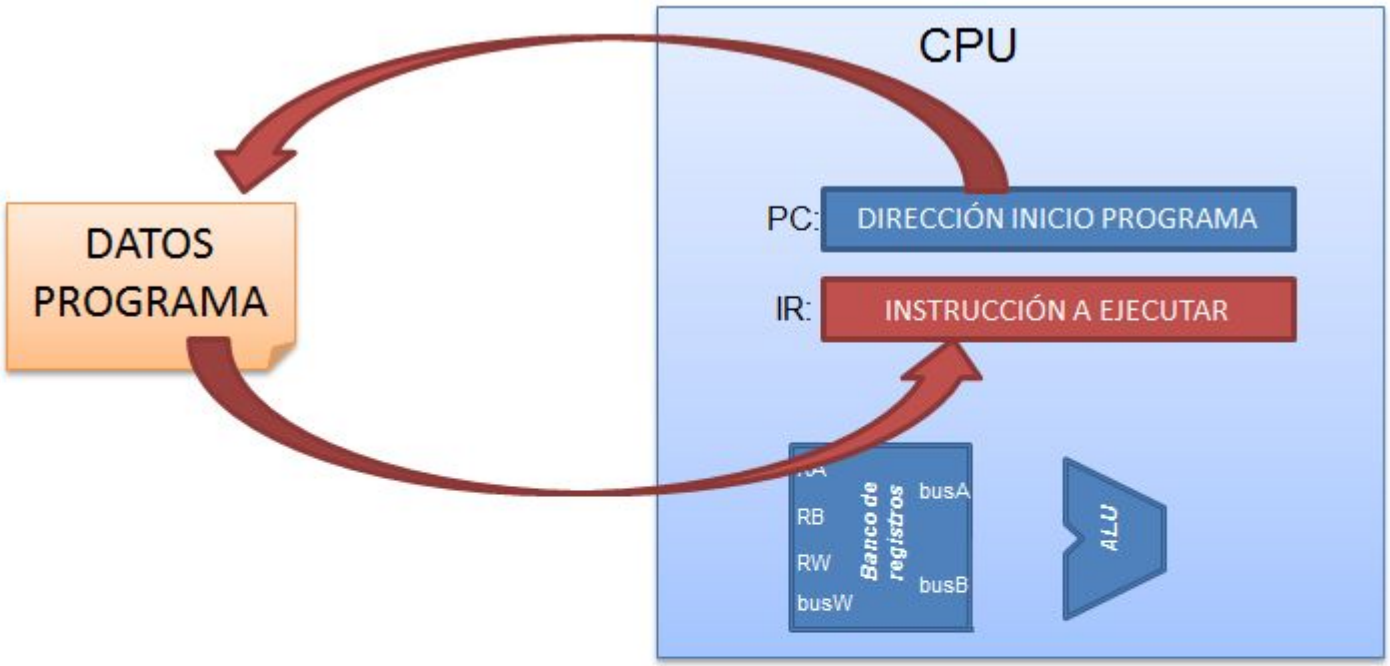
iii) El sistema operativo almacena el programa y sus datos en memoria RAM

Ejecución instrucción a instrucción



La dirección de memoria de la siguiente instrucción se encuentra en el registro PC

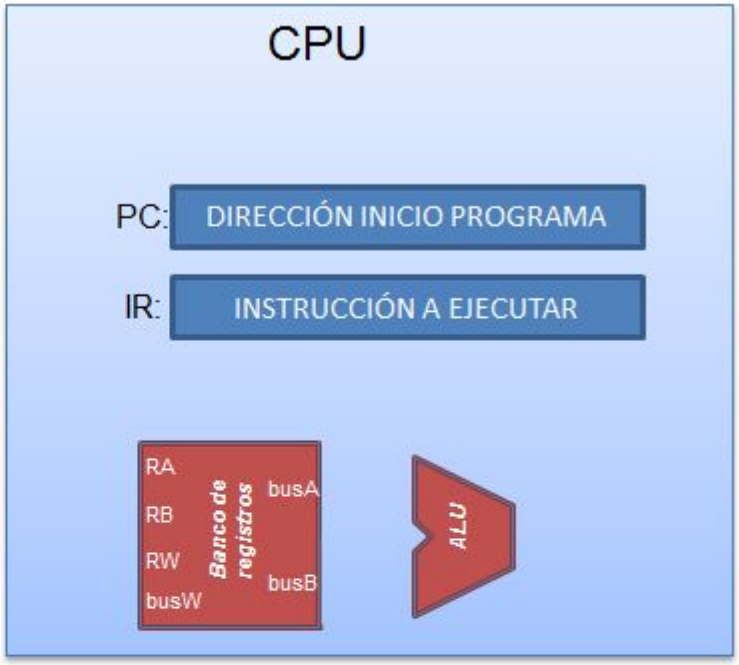
¿Cómo se ejecuta mi código?



La CPU busca la instrucción que tiene que ejecutar

Qué tiene que hacer
Dónde están los datos
Dónde se escribe el resultado

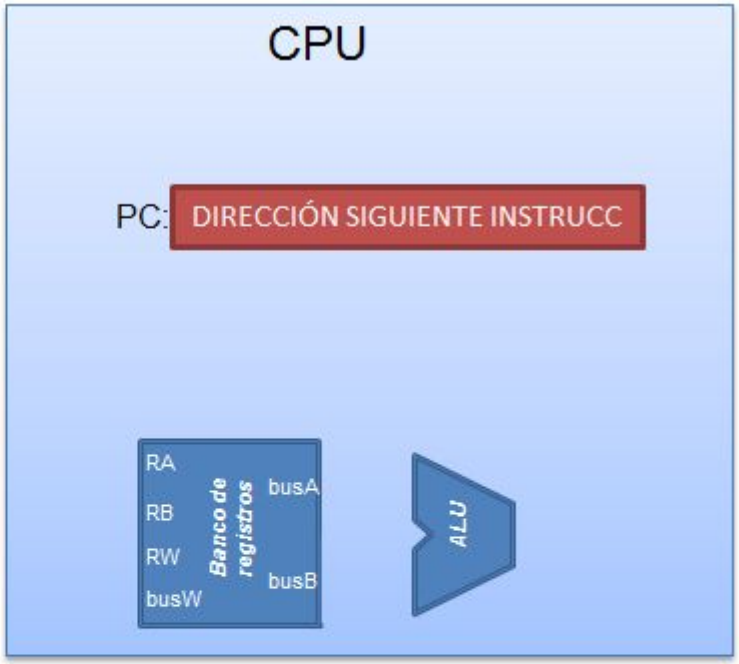
¿Cómo se ejecuta mi código?



La CPU ejecuta la instrucción

Leer los datos
Realizar la operación
Escribir el resultado

¿Cómo se ejecuta mi código?



La CPU calcula automáticamente dónde se encuentra la

Y ya está....
La CPU no da más de sí !!!

Pero entonces …

- ¿Cómo se lleva el fichero ejecutable a memoria?
  - ¡¡¡ Haciendo doble clic !!! Sí, claro.
- ¿A qué zona de la memoria se lleva?
- Una vez en memoria, ¿por qué/cómo empieza a ejecutarse mi código?
- El sistema operativo proporciona servicios para cargar un ejecutable en memoria y comenzar su ejecución

¿Y sólo puedo ejecutar una cosa?

- Mi sistema operativo (Windows no lo sé…) me permite ejecutar varios programas a la vez
  - Fácil, no hay más que repetir el proceso anterior *n veces…. o no.*
- Los *programas* deberán compartir mútiples recursos, empezando por la CPU
  - *"Es que mi ordenador tiene 4 cores"*
  - Y el mío, pero ahora mismo hay más de 100 *programas* ejecutándose *a la vez*

Gestión de procesos

- El sistema operativo ofrece la noción de ***proceso***
  - Un *proceso* es un programa **en ejecución**
  - Ejecuto mi código y, antes de que termine, lo vuelvo a ejecutar…. ¿cuántos procesos se crean?
- A grandes rasgos, el sistema operativo *reparte* el uso de CPU entre los procesos
  - Servicio de planificación

Procesos, dinamismo, memoria …

- Tu ordenador puede tener 4 cores, pero seguro que sólo tiene una memoria…
  - Otro recurso que deben compartir los procesos
- Cuándo decido ejecutar un programa (y se crea un proceso), ¿en qué posición de memoria escribo su código, datos…?
  - ¿Qué direcciones están libres/ocupadas?
    - Si una posición está a 0, libre. Si no, ocupado… Va ser que no.
  - ¿Y si el proceso A decide quedarse la memoria de B?

Gestión de memoria

- El sistema operativo se hace cargo de…
  - Gestionar el espacio libre
  - Asignar memoria a los nuevos procesos
  - Evitar accesos a zonas de memoria que pertenecen a un proceso
- Un momento, pero si hay más de 100 procesos ejecutándose a la vez….¿caben todos en memoria?
  - Lo más probable es que no
  - Más trabajo para el sistema operativo…

El mundo se vuelve paralelo

- Aparentemente, o nuestro código se paraleliza, o se queda obsoleto
  - ¿Varios hilos de ejecución cooperando para acelerar los cálculos? No suena fácil
  - Tendrán que esperarse y avisarse de eventos entre sí → ***sincronización***
  - Tendrán que **comunicarse** resultados
- El sistema operativo ofrece mecanismos de *comunicación y sincronización*
  - Entre procesos y entre hilos (ya veremos su diferencia)

Recapitulando …

- El sistema operativo nos proporciona servicios para…
  - Ejecución de programas, mediante la creación de procesos
  - Repartir el uso de la CPU y la memoria entre procesos
  - Establecer mecanismos de comunicación y sincronización entre procesos (e hilos)
- Vale. Pero además de eso, ¿qué hace el sistema operativo por nosotros?

La información organizada, por favor

- Constantemente creamos y borramos ficheros y carpetas, pero… ¿qué es un fichero?, ¿qué es una carpeta?
  - Son *datos* que están en el disco … ¿y?
- Quiero abrir el fichero "C:/tmp/paquito.txt" ¿Qué hay que hacer?
  - Doble click. Sí, claro
  - ¿Cómo sé en qué sector del disco comienza ese fichero?
- ¿Qué contiene un directorio?

Organizando la información

- El disco es como la memoria, pero a lo bestia
- Mismos problemas que para la gestión de memoria:
  - ¿Qué zonas están libres/ocupadas?
  - Al crear un nuevo fichero, ¿dónde lo coloco?
  - Dado un fichero, ¿cómo lo encuentro en el disco?
  - Si un fichero es muy grande, ¿debe estar consecutivo en el disco?
  - Ese fichero me pertenece; mecanismos de privacidad/protección

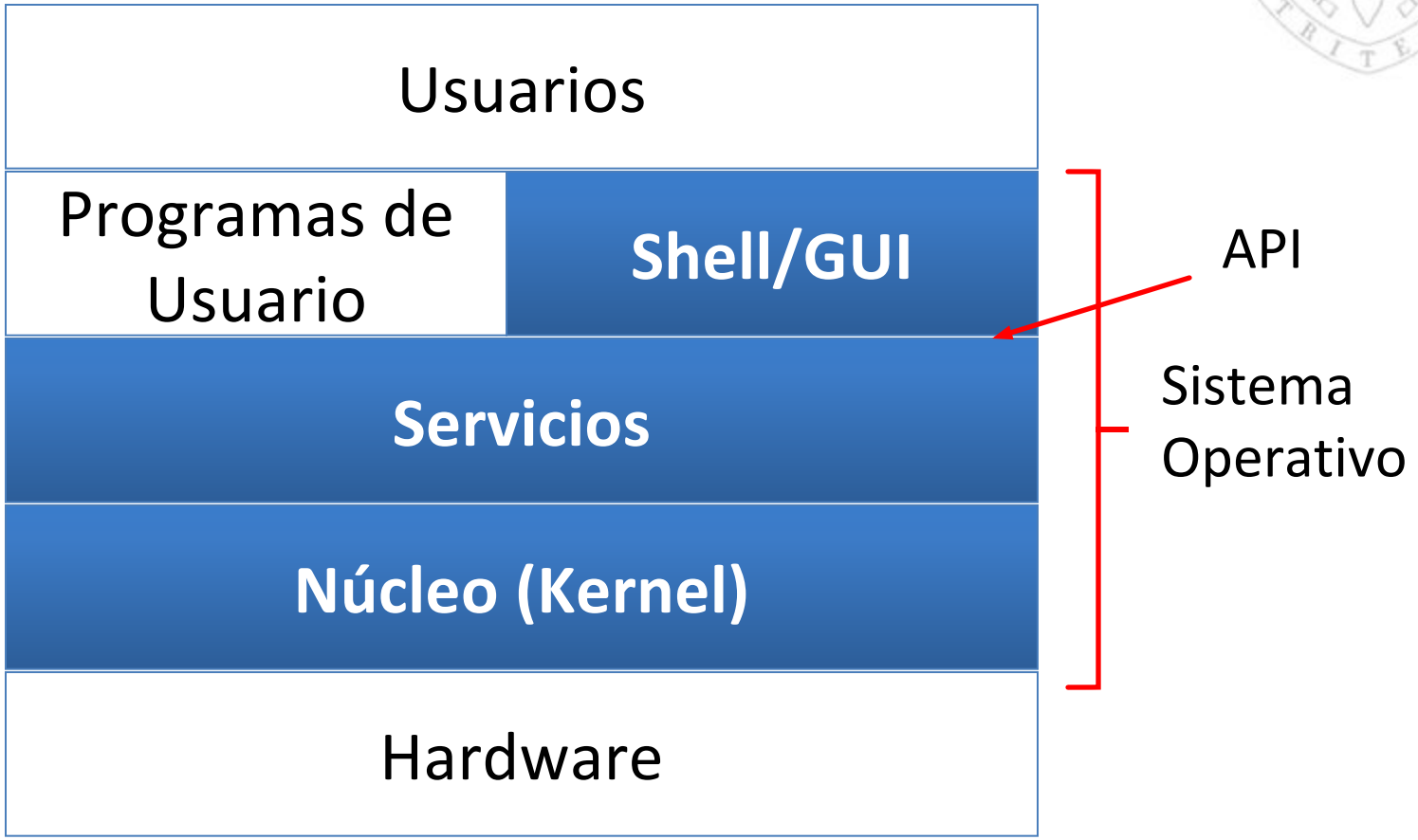
Sobre discos, impresoras y ratones

- ¿Y si usar un dispositivo de E/S supusiera revivir el laboratorio del ARM día tras día?
  - Programar dispositivos de E/S puede ser un poco tedioso: interrupciones, DMA….
- Definitivamente, que lo haga el sistema operativo.
  - Proporciona interfaces *friendly* para el acceso a los dispositivos
  - La funcionalidad ardua la realizar el *driver*
  - ¿Y cómo comunicar el *driver* con mi código?

Una de definiciones

- Pero entonces, ¿qué es un sistema operativo?
  a. Un hardware muy complejo que controla la CPU, memoria y el resto del hardware no tan complejo
  b. Un código escrito en ensamblador que se encontró junto a los manuscritos del Mar Muerto
  c. Miles de máquinas de estado jerárquicas escritas en VHDL ejecutándose en una FPGA junto a la CPU
  d. Código y sólo código, escrito por ejemplo en C, estrechamente relacionado con el hardware sobre el que se ejecuta
  e. Sólo hay dos personas en el mundo que sepan hacer un sistema operativo. Está fuera de las competencias de un informático.

Niveles del sistema operativo



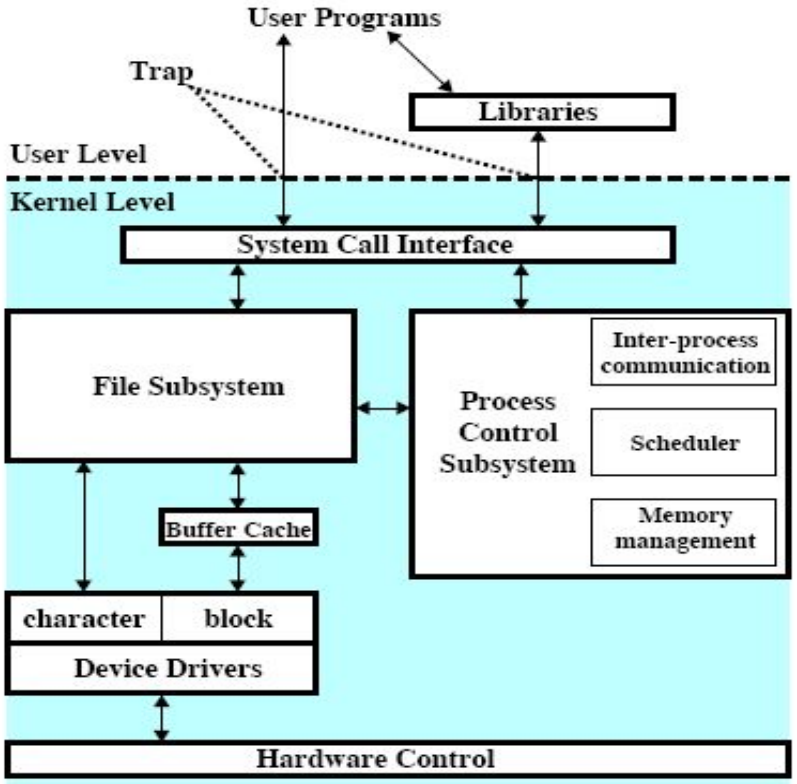
¿El sistema operativo es software?

- Pero si el sistema operativo es un *código más*… si se empieza a ejecutar el reproductor de vídeo con Ben-Hur, ¿cuándo/cómo puede ejecutarse código del sistema operativo?
  - Por invocación directa desde el código del reproductor: llamada al sistema
  - Por la llegada de una interrupción (un timer, por ejemplo)
  - Porque se produzca una excepción (el reproductor incurre en una *violación de segmento*, por ejemplo)

Servicios del sistema operativo, llamadas al sistema

- Es fácil confundir una llamada al sistema con una función cualquiera….
  - Desde el punto de vista del programador, son indistinguibles
- Pero no son iguales:
  - Dicha función no está en el espacio de direcciones del proceso sino del kernel
  - En SOs monolíticos (como GNU/Linux), la función debe ejecutarse completamente en modo kernel (modo privilegiado)
- Para invocar una llamada al sistema desde un proceso de usuario es preciso forzar un cambio de modo usuario a modo kernel (instrucción trap)

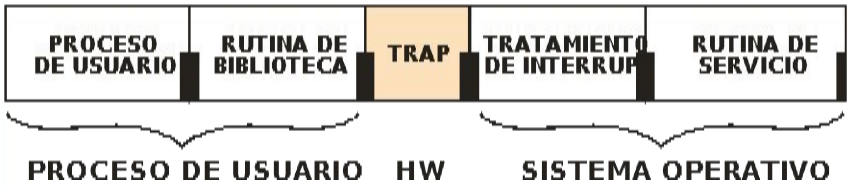
Llamadas al sistema: ejemplo




```
int main(void) {
  ...
  v=read(fd,buf,nbytes);
  ...
}
```

PASOS

1. Apilar* argumentos (fd,buf,nbytes)
2. Apilar* número de llamada al sistema (0 para read)
3. Ejecutar una instrucción *trap* que genera una interrupcion SW especial para cambiar de modo usuario a modo núcleo
4. El kernel accede a la tabla de llamadas al sistema para acceder a la función que la implementa y la invoca
5. Una vez finalizada la ejecución de la función, el kernel apila* el valor de retorno y vuelve a modo usuario



(*) Estos valores pueden pasarse usando registros, memoria o a través de la pila del programa

Activación de los servicios del sistema operativo (SO)

- El sistema operativo se activa también periódicamente → interrupciones generadas por el temporizador del sistema (cambio de modo)
  - Planificación: CPU accounting, expropiaciones de procesos/hilos, equilibrado de carga, …
  - Gestión del almacenamiento intermedio (escrituras a disco desde la cache de bloques)
  - Tareas periódicas asociadas a los protocolos de red

Estándar POSIX

- Interfaz estándar de sistemas operativos de IEEE.
- Objetivo: portabilidad de las aplicaciones entre diferentes plataformas y sistemas operativos.
  - POSIX: *Portable Operating System Interface for uniX*
- NO es una implementación. Sólo define una interfaz
- Diferentes estándares
  - 1003.1 Servicios básicos del SO
  - 1003.1a Extensiones a los servicios básicos
  - 1003.1b Extensiones de tiempo real
  - 1003.1c Extensiones de procesos ligeros
  - 1003.2 Shell y utilidades
  - 1003.2b Utilidades adicionales

Características de POSIX

- Nombres de funciones cortos y en letras minúsculas
  - `fork`
  - `read`
  - `close`
- Las funciones normalmente devuelven 0 en caso de éxito o –1 en caso de error
  - Variable `errno`
- Recursos gestionados por el sistema operativo se referencian mediante descriptores

El sistema operativo es SW… ¡pero necesita mucho HW!

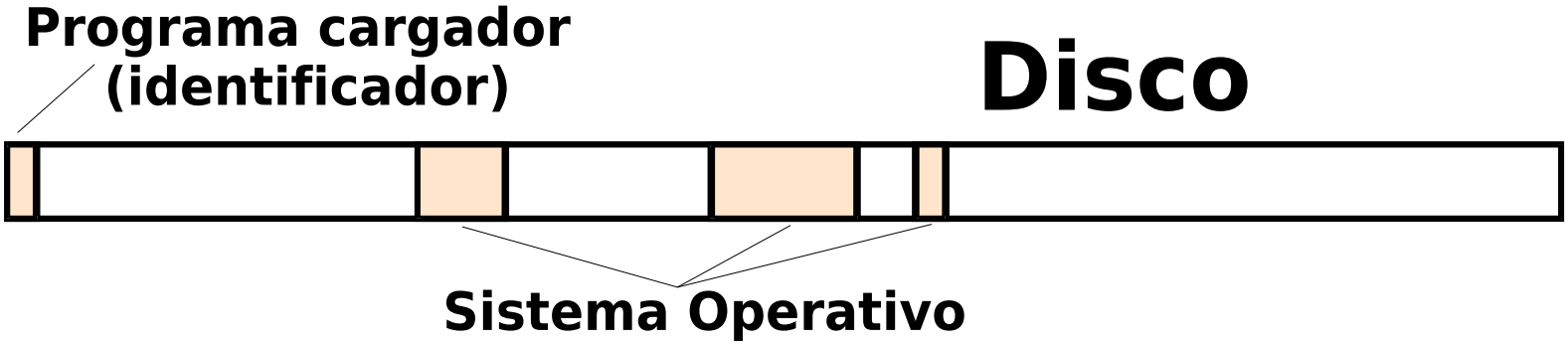
- Muchas funciones del SO que requieren soporte HW:
  - Permitir la ejecución de varios procesos, posiblemente multiplexando el uso de CPU
  - Proteger el acceso a memoria entre procesos
  - Permitir que dos o más procesos compartan memoria
  - Permitir a los procesos que utilicen los dispositivos de E/S, garantizando que el SO arbitre el uso de los mismos
  - Evitar que los procesos de usuario corrompan el código y las estructuras de control del SO
- Tres mecanismos HW esenciales:
  - Distintos modos de ejecución del procesador
  - Soporte HW para memoria virtual (MMU)
  - Temporizador del sistema (interrupciones periódicas)

Modos de ejecución del procesador

- El **procesador** ofrece un **conjunto de modos de ejecución** para proporcionar **distintos niveles de acceso a los recursos** de la máquina
- Cada modo de ejecución está caracterizado por:
  - Subconjunto de instrucciones del repertorio disponibles
  - Acceso al mapa de E/S
  - Acceso a los registros de soporte de gestión de memoria (config. MMU)
  - Permiso de cambio de modo (Bits del registro de estado)
- En GNU/Linux, **el núcleo del SO se ejecuta en el modo menos restrictivo** (“modo kernel”) y los **programas de usuario en el más restrictivo** ("modo usuario")
- Las arquitecturas x86 (Intel y AMD) ofrecen 4 niveles de privilegio (*ring levels*): 0, 1, 2 y 3.
  - Modo usuario: ring level 3
  - Modo kernel: ring level 0

¿Y cómo empieza todo?

- El SO está almacenado en el disco



- El computador incluye una memoria ROM (no volátil) que contiene un programa encargado de transferir a memoria el cargador del SO y ejecutarlo
  - *Iniciador ROM*

Arranque del computador

- Bajo control del iniciador ROM
  - •Test del HW
  - •Carga en memoria el 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
  - •Paso, si procede a modo Memoria Virtual
  - •Completa la carga del SO residente
  - •Habilita las interrupciones
  - •Crea el proceso de *login*
- Se entra en la fase normal de funcionamiento del SO. El SO se queda a la espera de que se produzca alguna interrupción