Sistemas Operativos

Universidad Complutense de Madrid

2020-2021

Módulo 1: Introducción a los Sistemas Operativos

Juan Carlos Sáez

¿Necesitamos un sistema operativo?

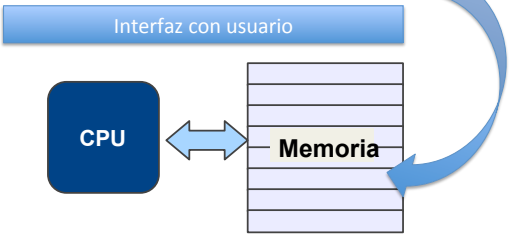
- A estas alturas ya sabemos...
  - Programar en lenguajes de alto nivel
  - Programar en ensamblador
  - Gestionar la E/S a bajo nivel
- ¿Qué ha hecho el sistema operativo por nosotros todo este tiempo?
  - Veámoslo como un *proveedor de servicios*

Repasemos lo que ya sabemos...

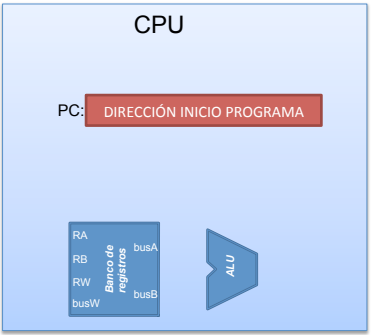
```
int main(void)
{
 int a, b, p;

 a = 5;
 b = 3;
 p = a * b;

 …
 return 0;
}
```

```
main:
   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
```

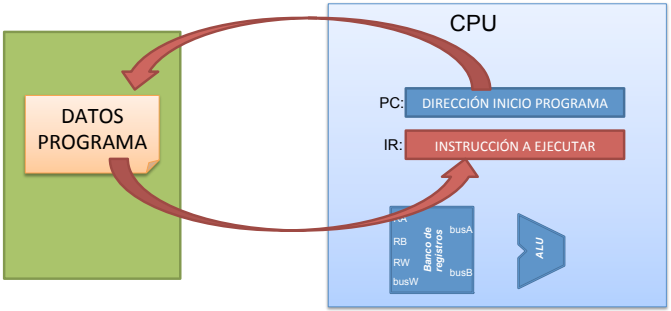


Ejecución instrucción a instrucción



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

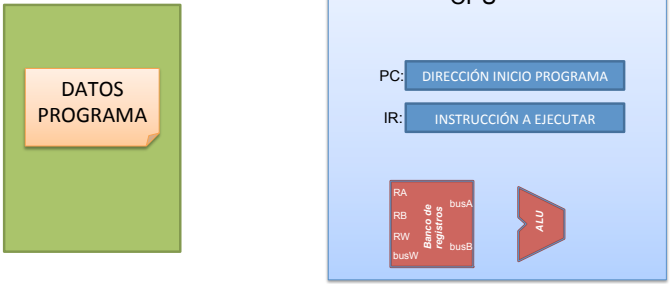
Ejecución instrucción a instrucción



La CPU busca la instrucción que tiene que ejecutar

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

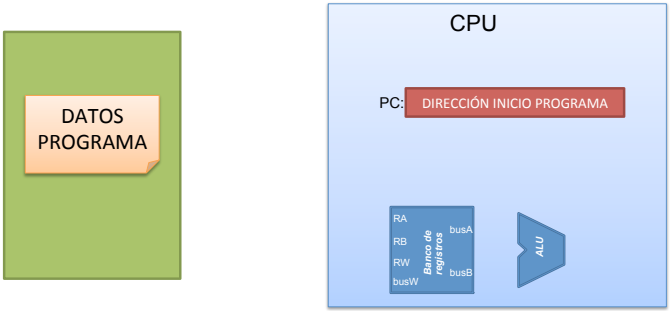
Ejecución instrucción a instrucción



La CPU ejecuta la instrucción

Leer los datos
Realizar la operación
Escribir el resultado

Ejecución instrucción a instrucción



La CPU calcula automáticamente dónde se encuentra la siguiente instrucción del programa

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 un programa?

- 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 programa 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
  - Planificador de procesos

Procesos, dinamismo, memoria...

- *"Tu ordenador puede tener 4 cores, pero seguro que sólo tiene una memoria..."*
  - Otro recurso que deben compartir los procesos
- Cuando decido ejecutar un programa (y se crea un proceso), ¿en qué posición de memoria escribo su código, datos...?
  - ¿Qué direcciones están libres/ocupadas?
  - ¿Y si el proceso A decide acceder a la memoria del proceso B?

Gestión de memoria

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

Comunicación y Sincronización

- El sistema operativo ofrece mecanismos de comunicación y sincronización
  - Entre procesos y entre hilos (ya veremos las diferencias)
- La proliferación de los procesadores multicore ha promovido la creación de aplicaciones multihilo
  - ¿Varios hilos de ejecución cooperando para acelerar los cálculos? No suena fácil
  - **Sincronización**: Los hilos tendrán que esperarse y notificar eventos
  - **Comunicación**: Los hilos pueden necesitar intercambiar información (p.ej., resultados intermedios)

Recapitulando...

El SO 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 o directorio?
  - Son datos que están en el disco...¿¿y??

Quiero abrir el fichero `C:\tmp\foo.txt` ...

- ¿Qué hay que hacer?
  - Doble clic. Sí, claro...
  - ¿Cómo sé en qué sector del disco comienza ese fichero?

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 identificado por su ruta, ¿cómo lo encuentro en el disco?
  - Si un fichero es muy grande, ¿deben estar sus datos consecutivos en el disco?
  - Ese fichero me pertenece; mecanismos de privacidad/protección

Sobre discos, impresoras y ratones

- ¿Y si gestionar dispositivos de E/S fuera labor del programador de aplicaciones?
  - Esta labor puede ser un poco tediosa:
    - Interrupciones
    - Gestión de DMA
    - ...
- Definitivamente, que lo haga el SO:
  - Proporciona interfaces *friendly* para el acceso a los dispositivos
  - Permite arbitrar el acceso a los dispositivos de E/S por parte de distintos procesos
  - La funcionalidad ardua la realiza el driver
  - ¿Y cómo se comunica el driver con las aplicaciones?

Programa

- **Módulo 1**: Introducción a los sistemas operativos
- **Módulo 2**: Sistemas de ficheros
- **Módulo 3**: Gestión de Procesos, planificación y sincronización
- **Módulo 4**: Gestión de entrada/salida
- **Módulo 5**: Gestión de memoria

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 la piedra Rosetta y que todo el mundo utiliza desde entonces

(C) Software y *sólo* software, escrito por ejemplo en C y ensamblador, estrechamente relacionado con el hardware sobre el que se ejecuta

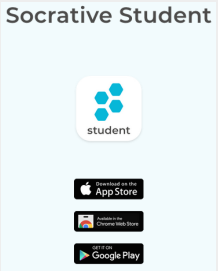
(D) La combinación adecuada de hardware y software para permitir la ejecución de aplicaciones de usuario en un computador

(E) Algo que está fuera de las competencias de un ingeniero informático. Sólo hay tres personas en el mundo que sepan hacer un sistema operativo.

Una de definiciones

1. Ir a **socrative.com**
2. Clic “Login” -> Clic “Student Login”
3. Introducir nombre de la sala (*Room Name*)
4. Clic “Join”

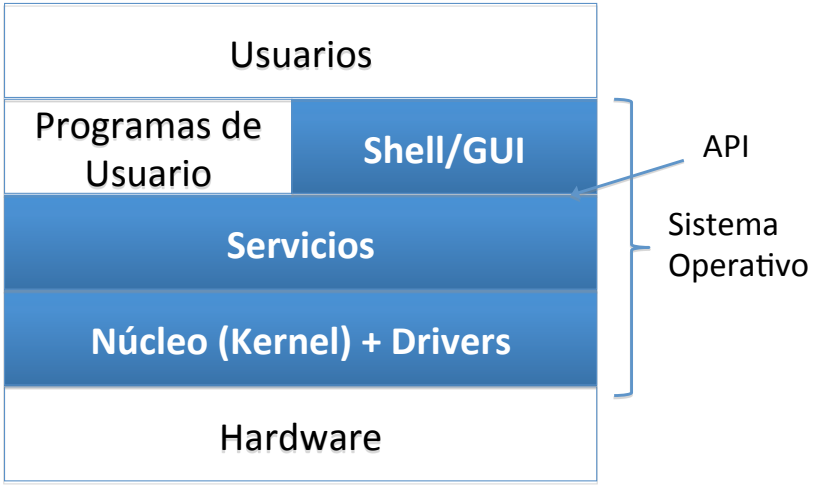
Alternativamente, puede usarse la *app*



Funciones del sistema operativo

- **Gestión de recursos** (CPU, memoria, ...)
  - Asignación y recuperación de recursos
  - Soporte de usuario y protección entre usuarios
  - Contabilidad/monitorización
- **Servicios** (*máquina extendida*)
  - Ejecución de programas (procesos)
  - Gestión de los dispositivos de E/S
  - Operaciones sobre ficheros y directorios
  - Detección y tratamiento de errores
- **Interfaz** de usuario
  - *Shell* o intérprete de comandos
  - GUI (*Graphical User Interface*)

Niveles del sistema operativo



Soporte hardware para el sistema operativo

- 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:

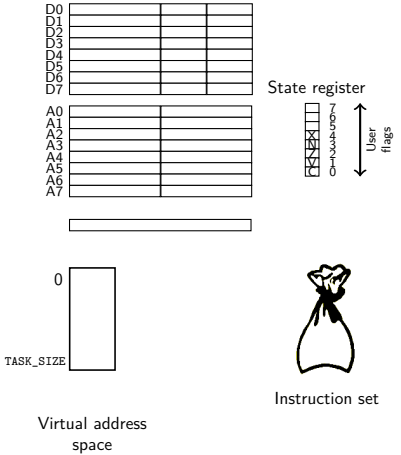
1. Distintos modos de ejecución del procesador
2. Soporte HW para memoria virtual (MMU - *Memory Management Unit*)
3. Temporizador del sistema (interrupciones periódicas)

Modos de ejecución del procesador

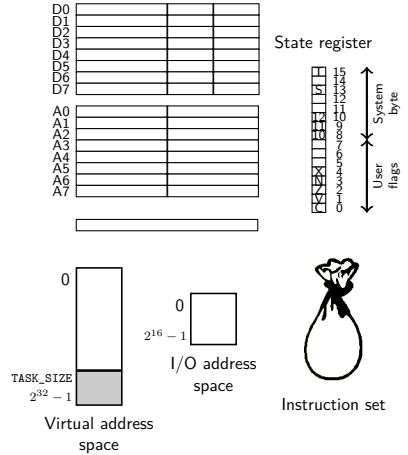
- Los procesadores actuales ofrecen un conjunto de modos de ejecución para proporcionar distintos niveles de acceso a los recursos de la máquina
- Cada modo de ejecución está caracterizado por:
  - Subconjunto de instrucciones del repertorio disponibles
  - Acceso al mapa de E/S
  - Acceso a los registros 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

Modo usuario vs. modo kernel

Modo usuario



Modo kernel



Activación del sistema operativo (I)

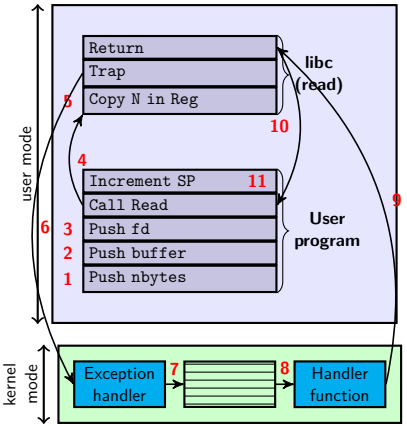
- Pero si el sistema operativo es simplemente software... *si se empieza a ejecutar el reproductor de vídeo con "Origen", ¿cuándo/cómo puede ejecutarse código del sistema operativo?*
- El SO se comporta como un *servidor*: sus funciones se ejecutan en respuesta a eventos
  1. Por invocación directa desde el programa de usuario (llamada al sistema)
     - Ej.: El reproductor solicita abrir y leer el fichero con la película
  2. Por la llegada de una interrupción (un temporizador, por ejemplo)
  3. Porque se produzca una excepción
     - Ej.: Hay un *bug* en el reproductor que desencadena una violación de segmento

Servicios del SO: llamadas al sistema

- Desde el punto de vista del programador, una llamada al sistema es *como cualquier otra función*
  - Las llamadas al sistema retornan un valor y pueden aceptar argumentos
- Pero en realidad, una llamada al sistema **NO** es una función convencional
  - La llamada provoca la invocación de una función que no está en el espacio de direcciones del proceso sino del kernel del SO
    - Un proceso no tiene acceso a ese espacio de direcciones
    - No basta usar una instrucción de salto para transferir el control al SO
  - En SOs monolíticos (como GNU/Linux), la función debe ejecutarse completamente en modo kernel (modo privilegiado)
- La invocación de una llamada al sistema desde un proceso de usuario fuerza un cambio de modo usuario a modo kernel
  - Instrucción especial del repertorio (TRAP)

Ejemplo de llamada al sistema

```
int main()
{
  char buffer[128];
  int count;
  int nbytes=128;
  int fd;

  ... Open the file ...

  count=read(fd,buffer,nbytes);

  ...
}
```



Estándar POSIX

- Interfaz estándar de sistemas operativos del 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 hilos
  - 1003.2: Shell y utilidades
  - 1003.2b: Utilidades adicionales

Características de POSIX

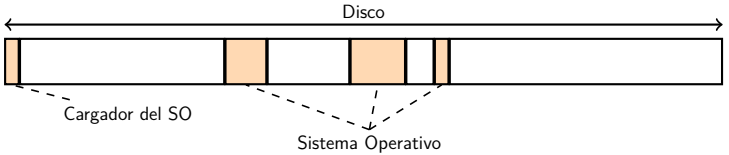
- Nombres de funciones cortos y en letras minúsculas
  - `fork()`
  - `read()`
  - `close()`
- Las funciones normalmente devuelven 0 en caso de éxito o –1 en caso de error
  - Código de error se almacena en la variable global `errno`
- Recursos gestionados por el sistema operativo se referencian mediante *descriptores*

Activación del sistema operativo (II)

- El sistema operativo se debe activar también periódicamente:
  - 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
  - ...
- Activación periódica del SO
  - Interrupciones generadas por el temporizador del sistema
  - Cada interrupción provoca una transición de modo usuario a modo kernel

¿Y cómo empieza todo?: Arranque del SO

- Los datos y el código del SO se almacenan en disco



- El computador incluye una memoria ROM (no volátil) que contiene un programa (Iniciador ROM) encargado de:
  1. Transferir a memoria el cargador del SO
  2. Ejecutar el cargador del SO (PC ← Dirección de la primera instrucción.)

Arranque del sistema

1. Iniciador ROM

- Test del HW
- Cargar en memoria el cargador del SO y transferir control

2. Cargador del SO

- Cargar en memoria los distintos componentes del SO

3. Sistema operativo

- Test del sistema de ficheros
- Creación de estructuras de datos internas (planificación, gestión de memoria, ...)
- Paso, si procede, a modo Memoria Virtual
- Cargar controladores de dispositivos (*drivers*)
- Habilitar interrupciones
- Crear proceso de *login*