# Mecanismos de entrada al Sistema

Sistemes Operatius – pla 2003 (SO)
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

## Licencia Creative Commons





## Licencia Creative Commons

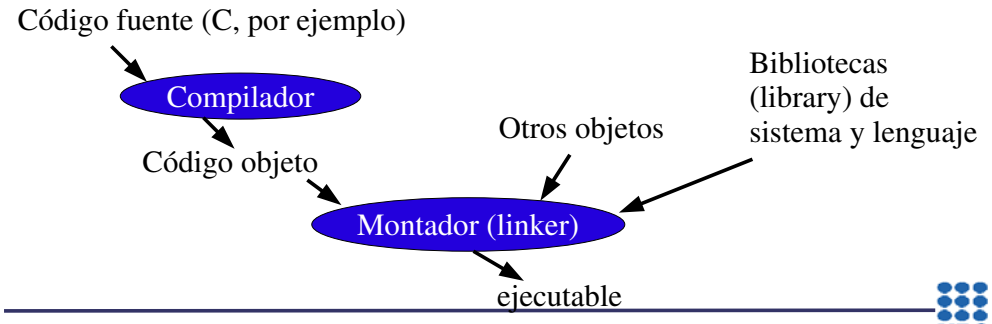




## Índice

## Introducción

- Un programa ejecutable se convierte en un proceso cuando el SO lo carga en memoria y le asigna un entorno de ejecución
- ¿De donde sale un ejecutable?





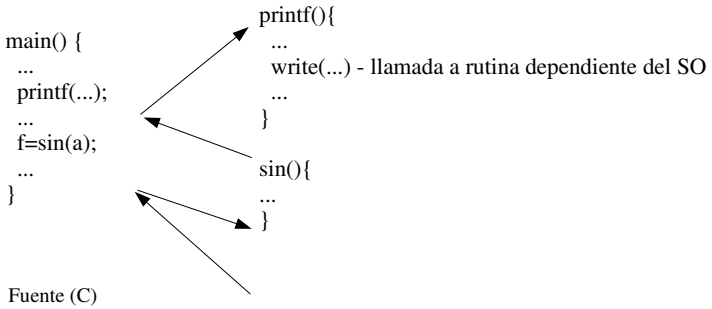

## Bibliotecas (mal llamadas librerias)

- Rutinas ya creadas, que se añaden al código y que el programador sólo necesita llamar
- De lenguaje y de sistema
- Permiten independencia
  - Un programa escrito en un lenguaje de alto nivel es independiente del SO y de la arquitectura en que se ejecutará
  - Una vez compilado y montado, está preparado para un SO y arquitectura



## Librerias de lenguaje

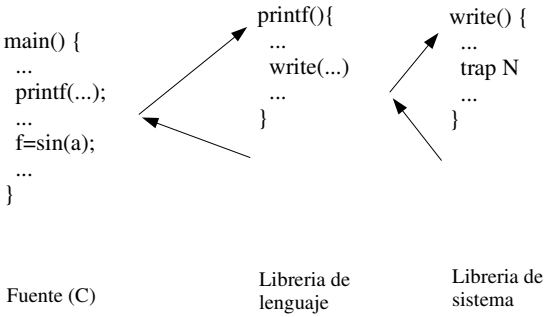
- Ligan un lenguaje con un SO (independientes de la arquitectura)
  - Algunas veces ya están autocontenidas (p. ej. cálculo del seno de una función). Entonces son independientes del SO.
  - Otras veces deben pedir servicios del sistema (p. ej. imprimir por pantalla)





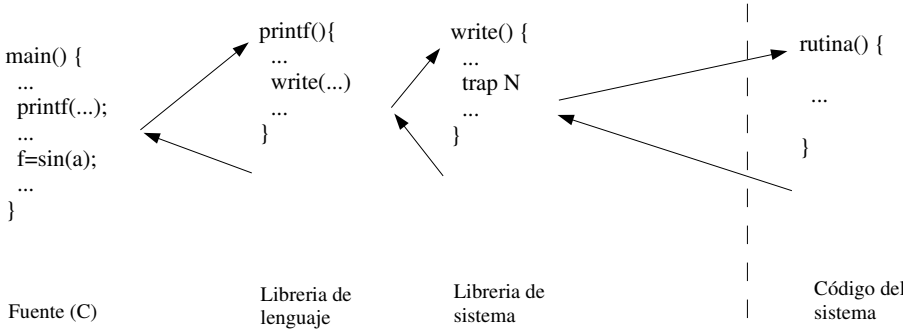

## Librerias de sistema

- Ligan un SO con una arquitectura (independientes lenguaje alto nivel)
- A su vez, el código de la libreria de sistema ejecuta código del SO (mecanismo *trap*)



## Código de sistema

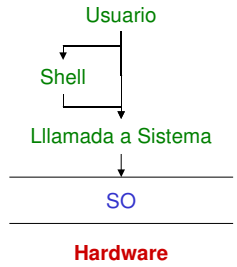
- El trap ejecuta código de sistema
  - Este código no forma parte del ejecutable
  - Hay que tener privilegios para ejecutar este código






## Las llamadas a sistema

- Un usuario realiza peticiones al SO mediante Llamadas a Sistema (*system calls*)
  - Estas llamadas quedan encapsuladas en las librerias (como ya hemos visto)
  - Pasamos de ejecutar código de usuario a código de sistema
  - El código de sistema debe estar protegido





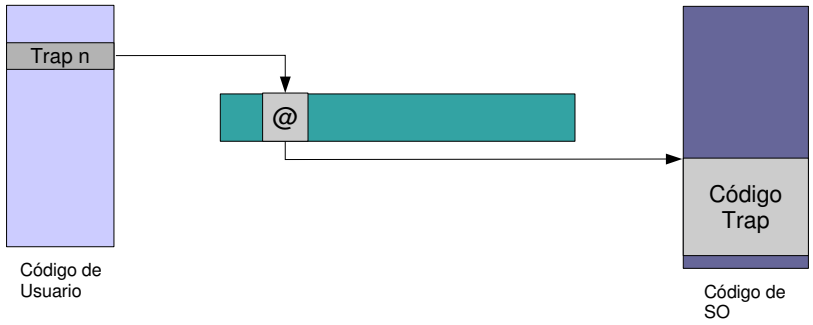

## ¿Cómo hacemos llamadas a sistema?

- Usar un CALL normal?
  - No, la @ del servicio puede variar
- *Inlining* del código de sistema?
  - Perdemos portabilidad
  - No podemos garantizar protección
- Necesitamos
  - Hacer llamadas a @ variables
  - Modos de ejecución para garantizar protección



## Soporte Hardware

- Permitir llamadas a @memoria variables
  - Utilizar una tabla de traducción
  - El SO inicializa las entradas de la tabla al iniciarse
  - Instrucción especial para este tipo de llamadas: TRAP



## Soporte Hardware (II)

- Modos de ejecución
  - Para garantizar protecciones el SO se ha de ejecutar con más privilegios que el resto de aplicaciones
  - Hace falta al menos 2 modos diferentes:
    - Modo no privilegiado
      - Juego de instrucciones limitado
      - Acceso a memoria limitado
    - Modo privilegiado ( sistema, kernel, supervisor )
      - Puede ejecutar todas las instrucciones del procesador
      - Puede acceder a toda la memoria
  - El mecanismo de TRAP cambia el modo de ejecución
    - Al entrar al SO pasa a modo privilegiado
    - Al salir el SO restaura el modo de usuario
  - Hay otros eventos que cambian de modo usuario – sistema



## Mecanismos de entrada

- Eventos que causan una interrupcion del flujo actual de usuario para realizar una tarea del SO
  - Interrupciones ( teclado, reloj, DMA, ... )
    - asíncronas
    - provocadas por dispositivos
    - entre 2 dos instrucciones de lenguaje máquina
  - Excepciones ( division por cero, fallo de página, ... )
    - síncronas
    - provocadas por la ejecución de una instrucción de lenguaje máquina
    - se resuelven dentro de la instrucción
  - Traps
    - provocados por una instrucción explicitamente
    - para pedir un servicio al SO (llamada al sistema)



## Mecanismos de entrada (II)

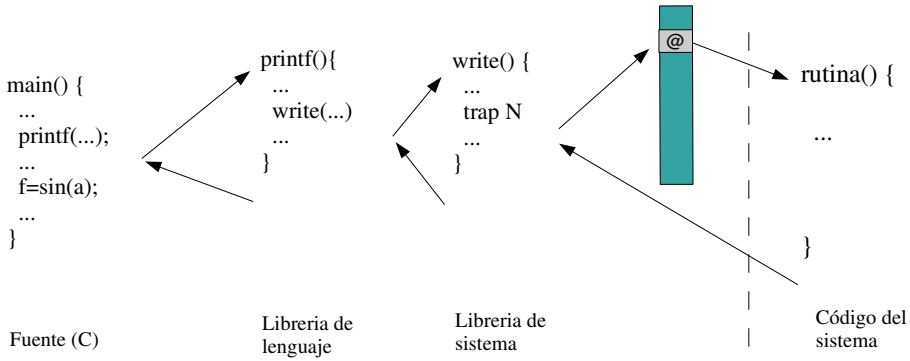
- Resumen

| | Síncrono o Asíncrono | Voluntario o Involuntario |
|---|---|---|
| Interrupciones | Asíncrono | Involuntario |
| Excepciones | Síncrono | Involuntario |
| **Traps** | Síncrono | Voluntario |



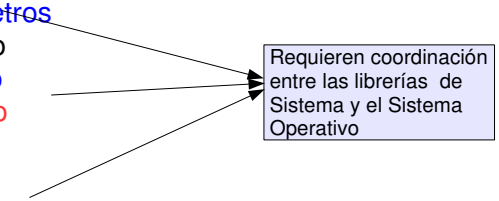
## La foto completa

- Las funciones de lenguaje usan llamadas a sistema
  - Están encapsuladas en librerias (lenguaje y sistema)
  - Usan el mecanismo trap, que cambia de modo de ejecución



## Soporte del SO

- Descripción de una Llamada a Sistema dentro del SO
  - Salvar contexto
  - Restaurar contexto sistema
  - Identificar servicio
  - Invocar el servicio
  - Recuperar parámetros
  - Realizar el servicio
  - Retornar resultado
  - Restaurar contexto

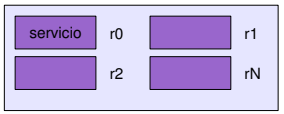





## Identificación del servicio

- 1 trap por servicio
  - Por cada servicio un trap diferente
  - No hay suficientes
- 1 trap por grupo de servicios
- 1 trap para todos los servicios
  - En ambos casos se especifica el servicio
    - **En un registro en el contexto de usuario**
      - Normalmente registro temporal
    - En una posición fija de memoria (pila)

```
trap ( )
{
   contexto = salvar();
   switch ( contexto->r0 ) {
      case N:
            Servei_N(); break;
   }
   restaurar ( contexto );
}

Servei_N ( )
{
}
```




Contexto de usuario



## Paso de parámetros

- 3 posibilidades
  - por registro
  - dejarlos en posiciones fijas de memoria (pila)
  - pasar un registro que apunte a los parámetros

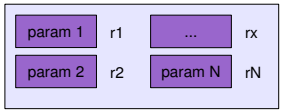


## Paso de parámetros (II)

- Usando registros
  - Cada parámetro en un registro
  - Rápido
  - Nº de parámetros limitado por el nº de registros

```
LlaS ( param1, param2, ... , paramN ) {
      salvar registros r1 a rN
      mov servicio, r0
      mov param1, r1
      mov param2, r2
      ...
      mov paramN, rN
      TRAP
      restaurar registros r1 a rN
}
```

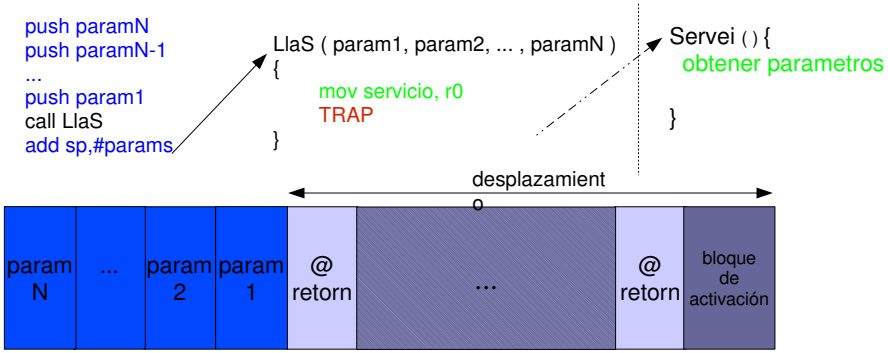
```
Servei ( ) {
  obtener parametros

}
```




Contexto de usuario

# Paso de parámetros (III)

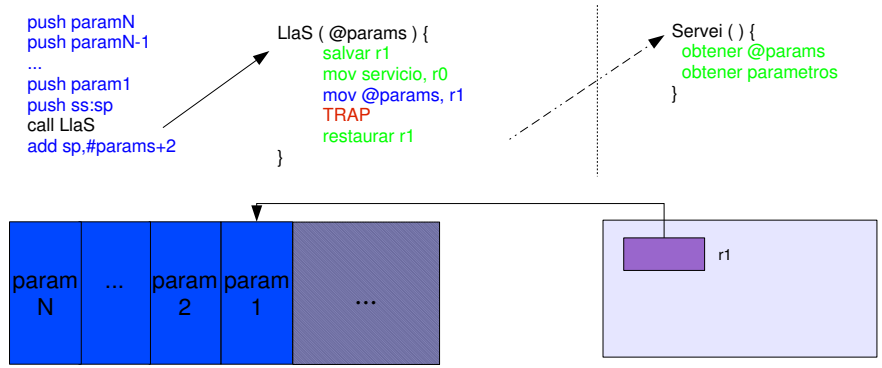
- Usando una zona fija de memoria
  - Normalmente en la pila
  - Utilizamos un desplazamiento fijo desde la base de la pila
  - Alto acoplamiento






# Paso de parámetros (IV)

- Usando un registro como apuntador
  - compromiso entre las dos anteriores






# Retorno de resultados

- Usando un registro
  - Se coloca el resultado en el contexto de usuario
- Usando una zona de memoria
  - Se deja el resultado en un lugar especifico de memoria ( pila )



# Ejemplo: i8086

- 1 modo de ejecución
  - SO no puede garantizar las protecciones
- Instrucción de Trap: INT n
  - Interrupción provocada por el software
  - Utiliza la tabla de interrupciones para la traducción
  - Salva la @ret y la PSW
- Al entrar en una interrupción
  - Interrupciones inhibidas automáticamente
  - Se pueden desinhibir explicitamente

## Ejemplo: i386

- 4 modos de ejecución
  - nivel 3 = modo usuario
  - nivel 2 = acceso a toda la memoria
  - nivel 1 = nivel 2 + algunas instrucciones privilegiadas
  - nivel 0 = todo
- Se cambia de modo si la rutina (servicio) que llamamos tiene un modo diferente
- En el i386 hay 4 modos (hardware), pero los SO no usan necesariamente todos



## Ejemplos: Soporte del SO

- Linux (i386):
  - Llamadas de Sistema agrupadas en la INT 80h
  - Paso de parámetros usando registros ( EBX, ECX, EDX, ESI, EDI )
  - Identificación de servicio y retorno de resultado con EAX
  - Usa los modos de ejecución 0 y 3 (sistema, usuario)
- Windows 2K (i386)
  - Llamadas de Sistema agrupadas en la INT 2eh
  - Paso de parámetros usando un puntero como registro (EDX)
  - Identificación de servicio y retorno de resultado con EAX
  - Usaba antiguamente los modos 0 y 3. A partir de NT4.0 usa 0,1,3, y a partir de W2K los 4 modos