T1-Introducción

SO-Grado 2011-2012

Índice

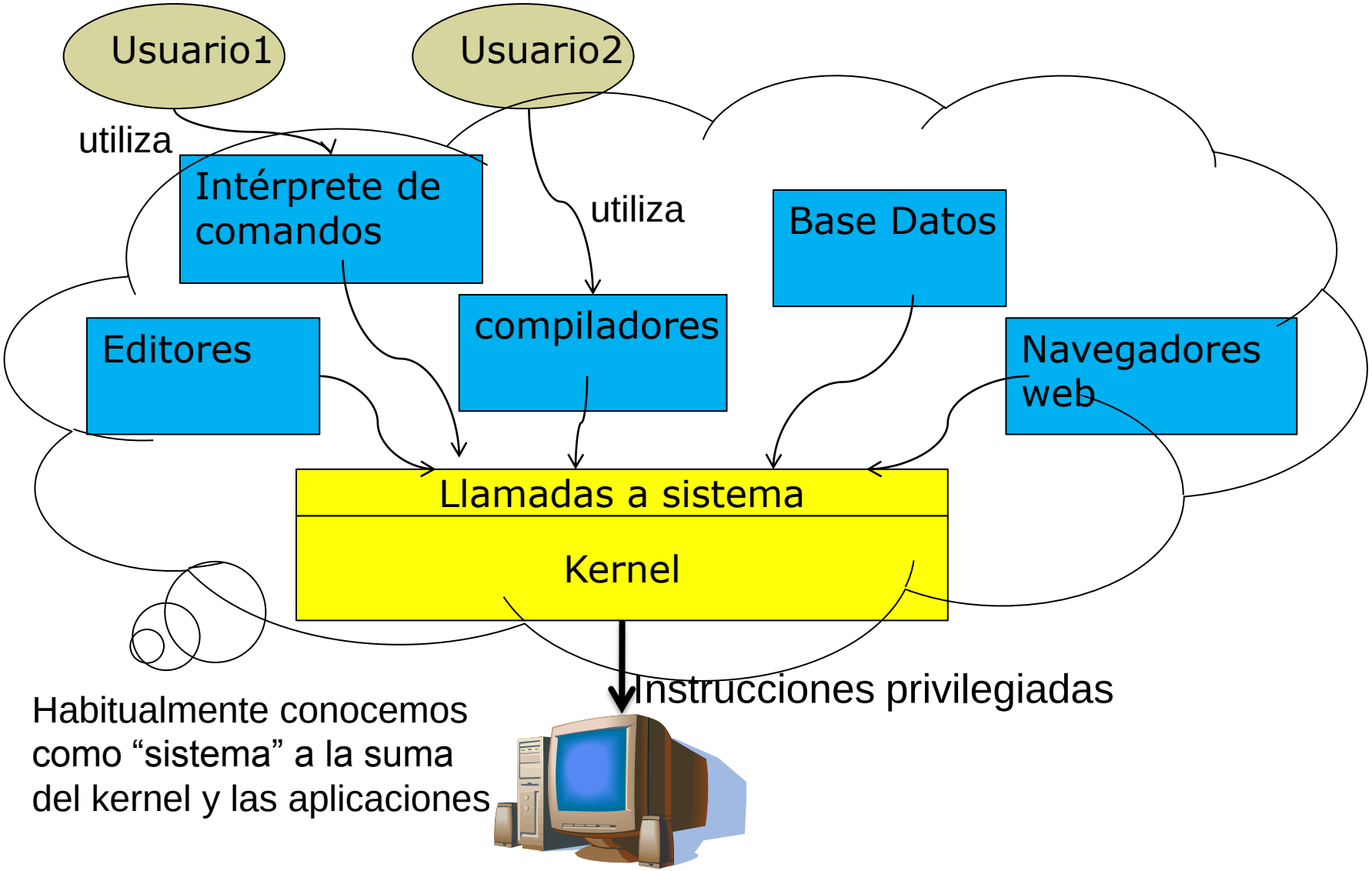
- El papel del S.O.
- Servicios que ofrece el S.O.
- Formas de acceder al kernel
  - Modos de ejecución
  - Interrupciones, excepciones y llamadas a sistema
- Llamadas a sistema
  - Generación de ejecutables
  - Requerimientos de las llamadas a sistema
  - Librerías

EL PAPEL DEL S.O.

¿Qué es un SO?

- El Sistema Operativo es un software que controla los recursos disponibles del sistema hardware que queremos utilizar y que actúa de intermediario entre las aplicaciones y el hardware
  - **Internamente** define estructuras de datos para gestionar el HW y algoritmos para decidir como utilizarlo
  - **Externamente** ofrece un conjunto de funciones para acceder a su funcionalidad (o servicios) de gestión de recursos

Componentes del sistema

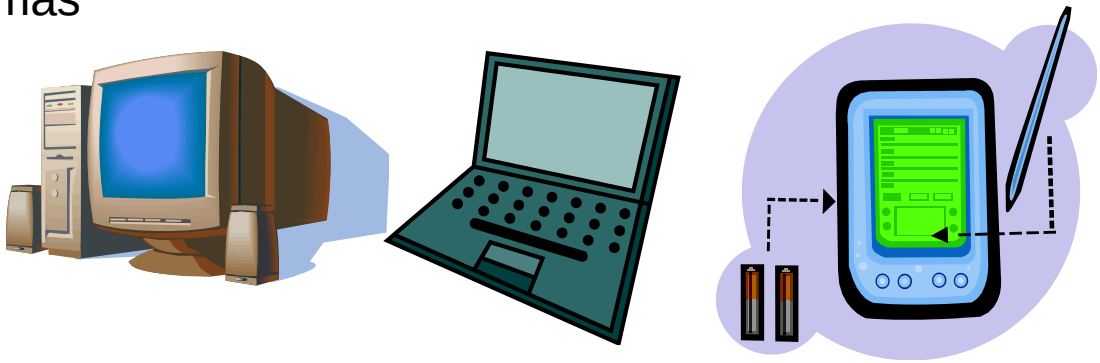


Habitualmente conocemos como “sistema” a la suma del kernel y las aplicaciones

SERVICIOS QUE OFRECE EL S.O.

¿Qué ofrece el S.O?

- Ofrece un **entorno "usable"**
  - Abstrae a los usuarios de las diferencias que hay entre los diferentes sistemas



- Ofrece un **entorno seguro**
  - Protege el HW de accesos incorrectos y a unos usuarios de otros
- Ofrece un **entorno eficiente**
  - Proporciona un uso eficiente de los recursos del sistema
  - Múltiples usuarios acceden a un mismo sistema y tienen una sensación de acceso en "exclusiva"

¿Qué ofrece el SO (en detalle)?

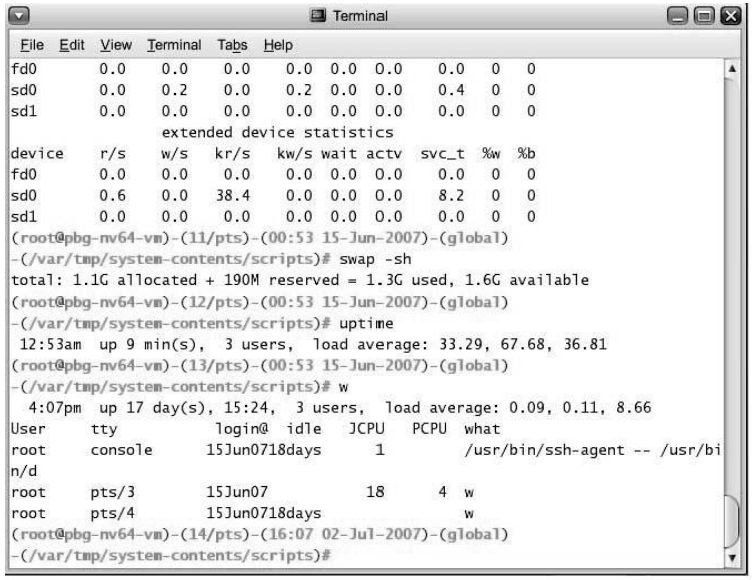
- Ofrece la inicialización y arranque del sistema → *Boot* o *startup*
- Ofrece un entorno de desarrollo de programas y de utilización de la máquina usable, seguro y eficiente
  - Aunque esto normalmente no forma parte del kernel sino que son programas externos
- Ofrece un conjunto de funciones (**llamadas a sistema**) para acceder a su funcionalidad de gestión de recursos
  - Ejecución de programas
  - Acceso a dispositivos (discos, red, cdrom, dvd, USB, etc)
  - Gestión de ficheros (como caso particular de acceso a disco)
  - Detección/Gestión de errores (software/hardware)
  - Registro de la utilización de los recursos (*accounting*): El sistema tiene límites de utilización de recursos
  - Protección: en cuanto a qué recursos puede utilizar cada usuario, de qué modo, qué operaciones pueden hacerse los programas entre ellos

Arranque del sistema

- El hardware carga el SO (o una parte) al arrancar. Se conoce como *boot*
  - El sistema puede tener más de un SO instalado pero sólo uno ejecutándose
  - El SO se copia en memoria (todo o parte de él)
  - Inicializa las estructuras de datos necesarias (hardware/software) para controlar la máquina i ofrecer los servicios
  - Las interrupciones hardware son capturadas por el SO
  - Se pone en marcha un *shell* (puede ser un entorno gráfico) que nos permite acceder de forma interactiva a los servicios del sistema

Entorno de desarrollo

- El SO ofrece, como parte de sus servicios, un entorno de trabajo **interactivo**. Fundamentalmente puede ser de dos tipos: intérprete de comandos o entorno gráfico
- Dependiendo del sistema, este entorno puede formar parte del *kernel* o puede ser un programa aparte. En cualquier caso lo encontramos como parte del sistema para poder utilizarlo.



Entorno “usable”, seguro y eficiente

- Gestión de CPU (Tema 2)
  - El sistema decide qué programas se ejecutan en cada momento, modifica los registros de la CPU para que se ejecuten y decide cuánto tiempo
  - Se ofrecen llamadas a sistema para poner programas en ejecución (procesos), pararlos, etc
- Gestión memoria (Tema 3)
  - El sistema carga los ejecutables en memoria, decide qué partes del programa pone en memoria, y qué zonas de la memoria asigna a cada programa
  - Se ofrecen llamadas a sistema para asignar más/menos memoria, compartir memoria entre programas, etc
- Gestión dispositivos E/S (Tema 4)
  - Existen muchos y variados tipos de dispositivos. El SO aísla a los programas de usuarios de las dificultades de acceder a los dispositivos y protege el acceso (sólo se pueden acceder en modo privilegiado)
  - Se ofrecen llamadas a sistema para acceder a los dispositivos para leer, escribir, etc
- Gestión Sistema de ficheros → (Tema 5)
  - Los ficheros de datos permiten almacenar información de forma persistente. Como organizar esa información es responsabilidad del sistema de ficheros. Incluye la gestión del disco que es un dispositivo de almacenamiento persistente
  - Se ofrecen llamadas a sistemas propias de los sistemas de ficheros

FORMAS DE ACCEDER AL KERNEL

Como ofrecer un entorno seguro: “Modos” (o estados) de ejecución

- El SO necesita una forma de garantizar su seguridad, la del hardware y la del resto de procesos
- Necesitamos instrucciones de lenguaje máquina privilegiadas que sólo puede ejecutar el kernel
- El HW conoce cuando se está ejecutando el kernel y cuando una aplicación de usuario. Hay una instrucción de LM para pasar de un modo (o estado) a otro.
- **El SO se ejecuta en un modo de ejecución privilegiado.**
  - **Mínimo 2 modos**
    - **Modo de ejecución NO-privilegiado , *user mode***
    - **Modo de ejecución privilegiado, *kernel mode***
  - **Hay partes de la memoria sólo accesibles en modo privilegiado y determinadas instrucciones de lenguaje máquina sólo se pueden ejecutar en modo privilegiado**

- Objetivo: Entender que son los modos de ejecución y porqué los necesitamos

Modos (o estados) de ejecución

- **El hardware proporciona un mecanismo para pasar de un modo a otro**
  - **Normalmente es una instrucción especial**
  - ¿Como sabe el HW en que modo está? En la palabra de estado hay información que indica el modo en que está ejecutando
    - Como mínimo 1 bit (*mode bit*), pero depende de cuantos modos de ejecución ofrezca el procesador (pentium, 4 modos, 2 bits)
    - Cuando se accede al kernel el hardware cambia de modo y al volver a usuario se cambia a modo usuario

¿Cuando se ejecuta código de kernel?

- Cuando una aplicación ejecuta una llamada a sistema
- Cuando una aplicación provoca una excepción
- Cuando un dispositivo provoca una interrupción

- Estos eventos podrían no tener lugar, y el SO no se ejecutaría → El SO perdería el control del sistema.
- **El SO configura periódicamente la interrupción de reloj para evitar perder el control y que un usuario pueda acaparar todos los recursos**
  - **Cada 10 ms por ejemplo**
  - **Típicamente se ejecuta la planificación del sistema**

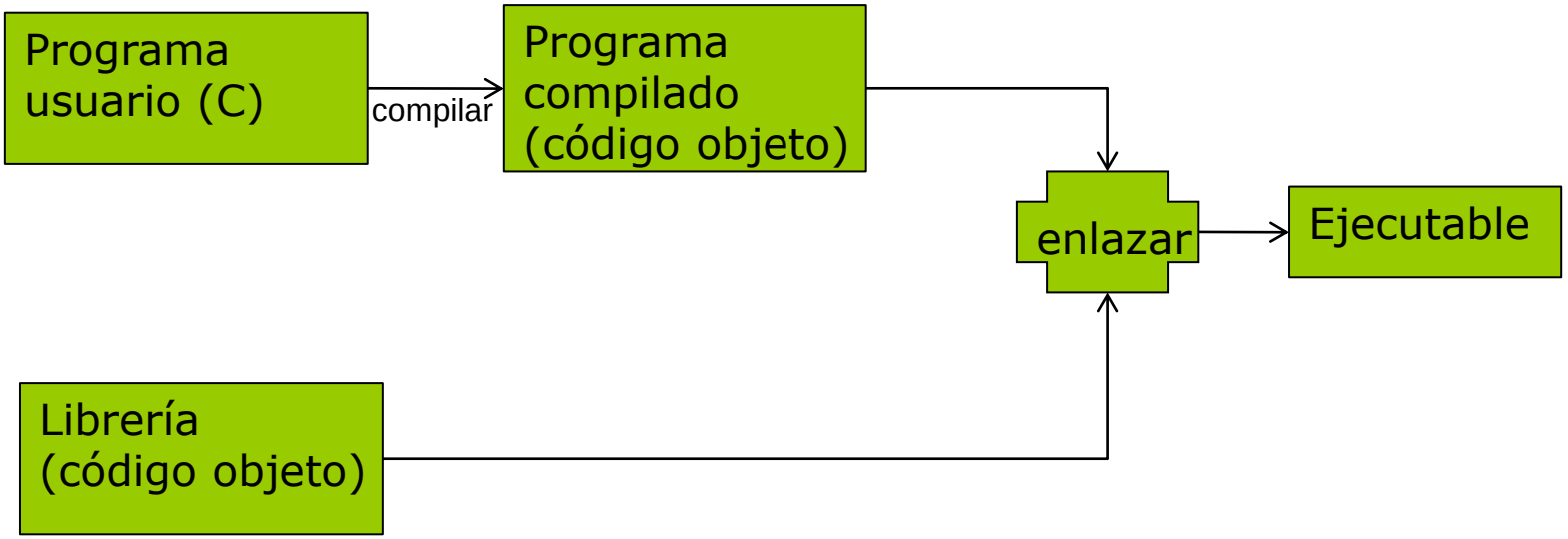
Acceso al código del kernel

- El kernel es un código guiado por eventos
  - Interrupción del flujo actual de usuario para realizar una tarea del SO
- Tres formas de acceder al código del SO (Visto en EC):
  - **Interrupciones** generadas por el hardware (teclado, reloj, DMA, ...)
    - asíncronas (entre 2 dos instrucciones de lenguaje máquina)
  - Los errores de software generan **excepciones** (división por cero, fallo de página, ...)
    - síncronas
    - provocadas por la ejecución de una instrucción de lenguaje máquina
    - se resuelven (si se puede) dentro de la instrucción
  - Peticiones de servicio de programas: **Llamada a sistema**
    - síncronas
    - provocados por una instrucción explícitamente (de lenguaje máquina)
    - para pedir un servicio al SO (llamada al sistema)
  - Desde el punto de vista hardware son muy parecidas (casi iguales)

- Generación de ejecutables
- Requerimientos de las llamadas a sistema
- Librerías

LLAMADAS A SISTEMA

Generación ejecutables



- Librerías: Rutinas/Funciones ya compiladas (es código objeto), que se "enlazan" con el programa y que el programador sólo necesita llamar
- Pueden ser a nivel de lenguaje (libc) o de sistema (libso)

Llamadas a Sistema

- Conjunto de funciones que ofrece el kernel para acceder a sus servicios
- **Desde el punto de vista del programador es igual al interfaz de cualquier librería del lenguaje (C,C++, etc)**
- Normalmente, los lenguajes ofrecen un API de más alto nivel que es más cómoda de utilizar y ofrece más funcionalidades
  - Ejemplo: Librería de C: printf en lugar de write
    - Nota: La librería se ejecuta en **modo usuario** y no puede acceder al dispositivo directamente
- Ejemplos
  - Win32 API para Windows
  - POSIX API para sistemas POSIX (UNIX, Linux, Max OS X)
  - Java API para la Java Virtual Machine

Requerimientos llamadas a sistema

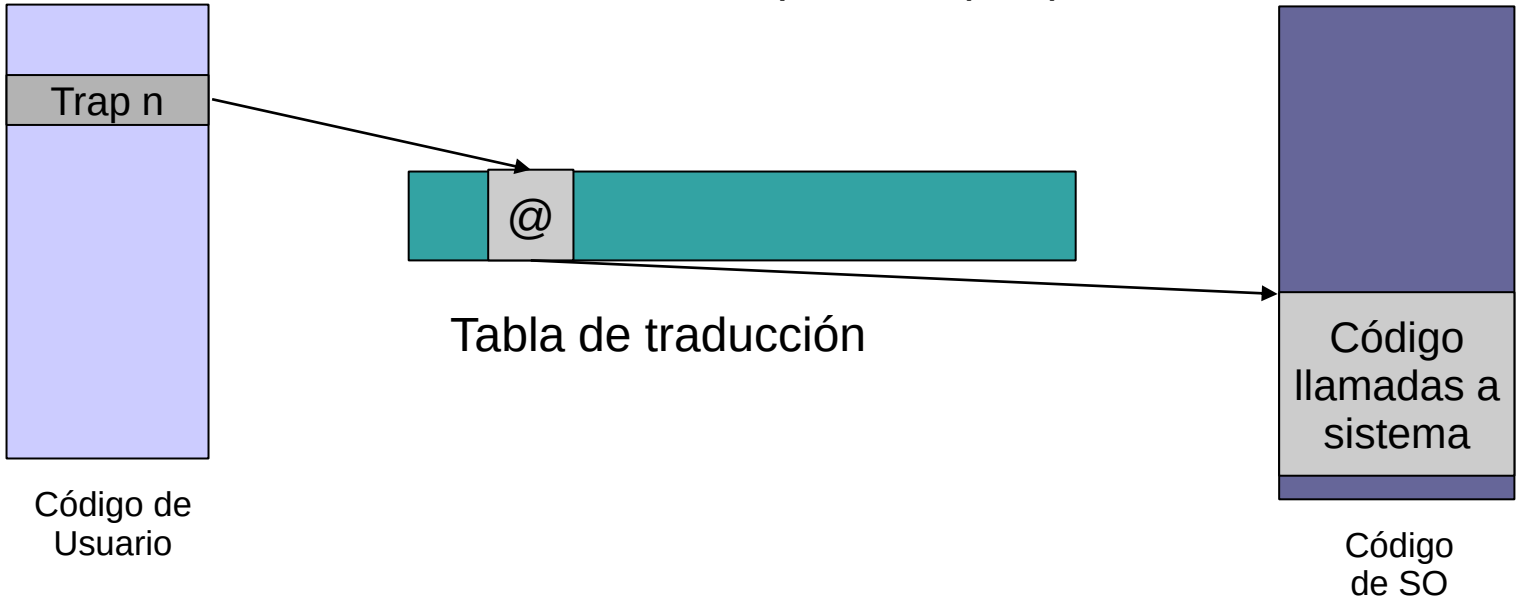
- Requerimientos
  - Desde el punto de vista del programador
    - Tiene que ser tan sencillo como una llamada a función
    - No se puede modificar su contexto (igual que en una llamada a función)
  - Desde el punto de vista del kernel necesita:
    - Ejecución en modo privilegiado
    - Paso de parámetros y retorno de resultados entre modos de ejecución diferentes
    - Las direcciones que ocupan las llamadas a sistema tienen que poder ser variables para soportar diferentes versiones de kernel y diferentes S.O.

Alternativas para la invocación de la llamada a sistema

- Usar un CALL normal?
  - No, la @ del servicio puede variar entre versiones del kernel
  - No podríamos ofrecer protección
- Inlining del código de sistema?
  - Perdemos portabilidad
  - No podemos garantizar protección
- **Sysenter, int, o similar. Le llamamosTrap para generalizar**
  - Instrucción de lenguaje máquina que **provoca el cambio de modo** de forma atómica y el **salto al código de sistema**
  - **Soporte hardware** (varía según la arquitectura)
  - Hay que poder pasar información adicional (paso de parámetros)
  - Hay que poder recoger un resultado (valor de retorno)

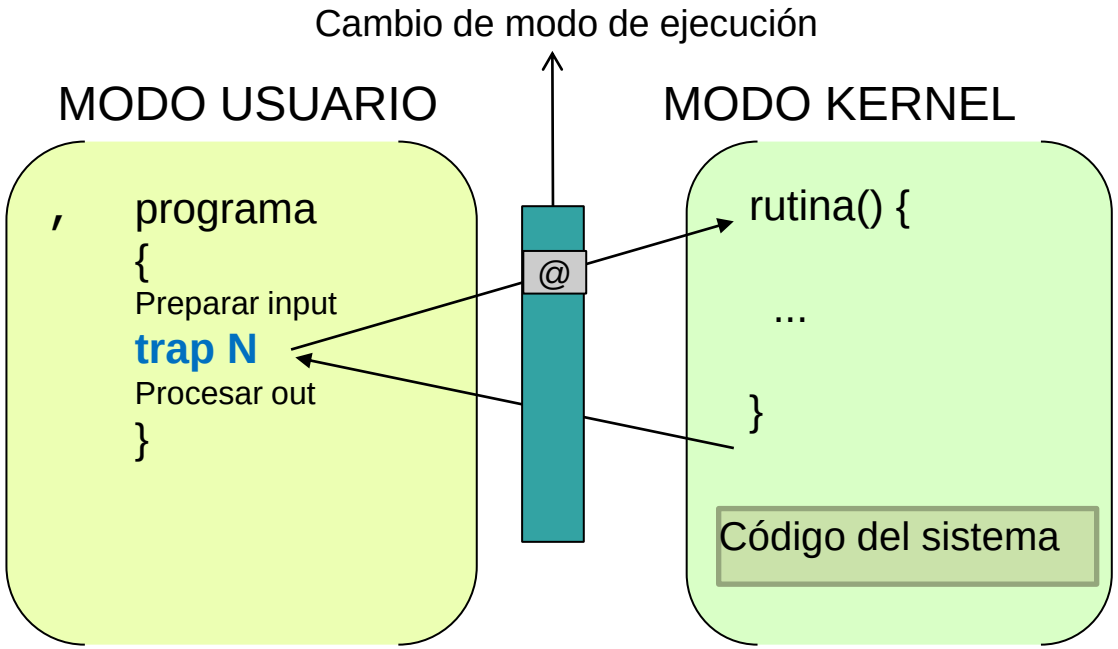
Soporte Hardware

- Instrucción especial **Trap**
  - En el intel, es la instrucción INT
- ¿Cómo permitir llamadas a @memoria variables?
  - Mediante una tabla de traducción.
    - El SO inicializa las entradas de la tabla al iniciarse (*boot*)
  - Parámetro del trap: índice de la tabla de traducción
  - En el intel, vector de interrupciones (IDT)



Código de sistema

- El código de kernel que se ejecuta no forma parte del ejecutable del programa de usuario



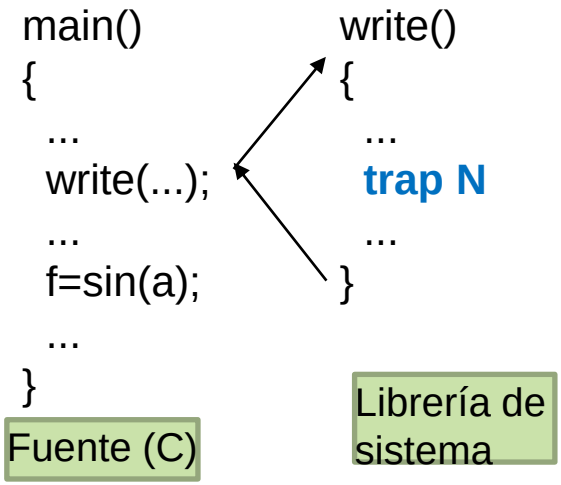
- Código de usuario de bajo nivel: **dependiente** de la **arquitectura** y de la versión del **SO → Sin embargo, queríamos que fuera fácil y esto no es fácil. Por ello, usaremos librerías que nos facilitará la invocación de las llamadas a sistema**

Librerías

- Existen diferentes tipos de librerías que facilitan la programación
  - Librerías de sistema: contienen la parte compleja de la invocación, paso de parámetros, etc de las llamadas a sistema. Las ofrecen los sistemas por defecto con su instalación. Aunque se llamen “de sistema” forman parte del ejecutable de las aplicaciones y por lo tanto es código ejecutado en modo usuario.
  - Librerías de lenguaje: Ofrecidas por los lenguajes de programación. Ofrecen funciones de uso frecuente. Algunas usan las funciones de las librerías de sistema para ofrecer funcionalidad más compleja y por lo tanto más útil.

Librerías “de sistema”

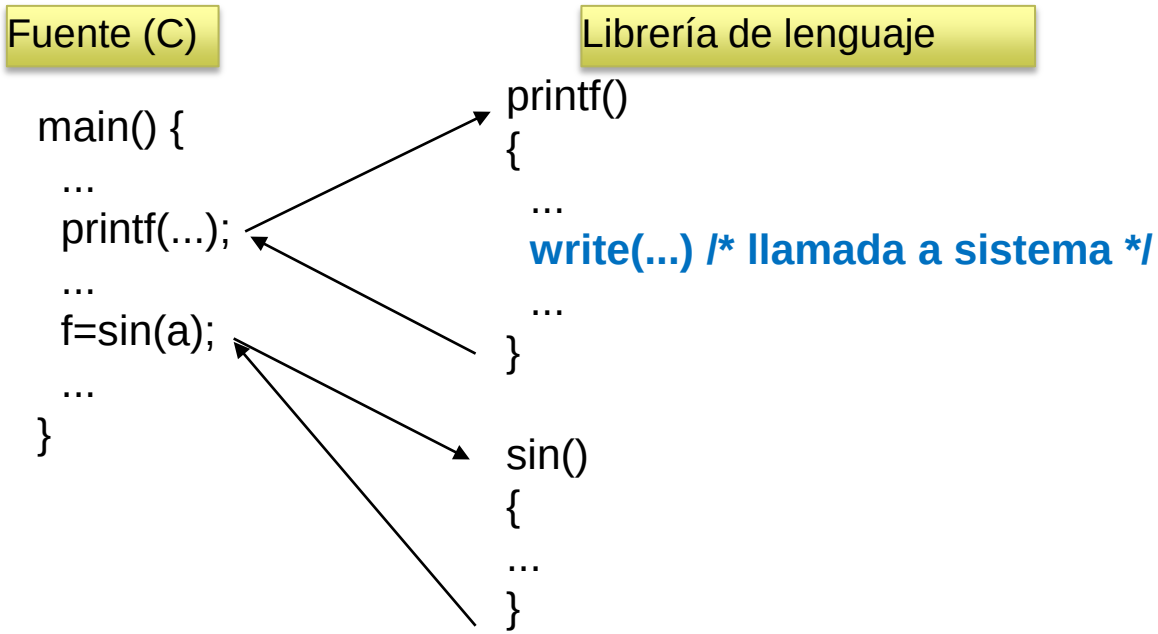
- El SO ofrece librerías del sistema para aislar a los programas de usuario de todos los pasos que hay que hacer para
  1. Pasar los parámetros
  2. Invocar el código del kernel
  3. Recoger y procesar los resultados



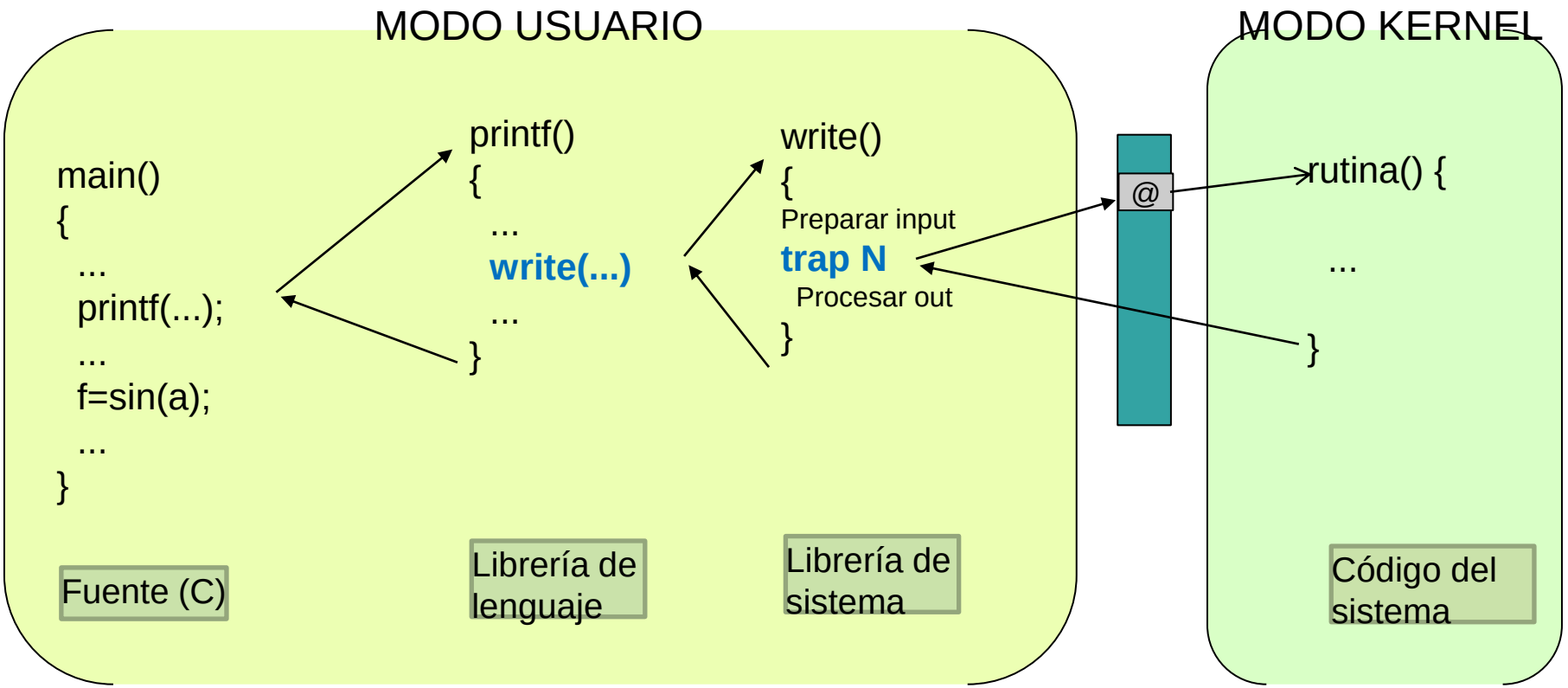
- Se llaman librería de sistema pero se ejecuta en **modo usuario, solamente sirven para facilitar la invocación de la llamada a sistema**

Librerías “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)
  - Se ejecutan en **modo usuario**



La foto completa



Código genérico al entrar/salir del kernel

- Hay pasos comunes a interrupciones, excepciones y llamadas a sistema
- En el caso de interrupciones y excepciones, no se invoca explícitamente ya que genera la invocación la realiza la CPU, el resto de pasos si se aplican.

	Función "normal"	Función kernel
Pasamos los parámetros	Push parámetros	**DEPENDE**
Para invocarla	call	sysenter, int o similar
Al inicio	Salvar los registros que vamos a usar (push)	Salvamos todos los registros (push)
Acceso a parámetros	A través de la pila: Ej: mov 8(ebp),eax	**DEPENDE**
Antes de volver	Recuperar los registros salvados al entrar (pop)	Recuperar los registros salvados (todos) al entrar (pop)
Retorno resultados	eax (o registro equivalente)	**DEPENDE**
Para volver al código que la invocó	ret	sysexit, iret o similar