Sistemas Operativos

Introducción

Qué es un Sistema Operativo

Agenda

Agenda

Recordemos: modelo von Neumann

- Un computador es un sistema digital compuesto de los siguientes módulos:
  - CPU (procesador)
  - Memoria
  - E/S
  - Interconexión (bus/red)
- La CPU *manda*:
  1. Traer la *siguiente* instrucción de la Memoria
  2. Decodificar la instrucción
  3. Obtener los operandos
  4. Realizar la operación indicada
  5. Guardar el resultado
  6. Volver al punto 1
- Observaciones:
  - Las instrucciones deben estar en memoria
  - Los datos de entrada deben estar en memoria
  - Los datos de salida se generan en memoria

Qué más sabemos sobre un computador

- Sabemos programar un computador en lenguaje de alto nivel
  - C, C++, Java, etc.
  - Sabemos que debe ser traducido a lenguaje máquina
- Stack: pila de llamadas para la implementación de funciones, metodos, etc
- Heap: Podemos pedir memoria dinámica al sistema
- Hay otras regiones de memoria: código, datos, ...
- Hemos manejado algunos dispositivos de E/S sin soporte de SO

La misión del SO es permitir al programador y al usuario un manejo simplificado de esta compleja máquina facilitando las tareas que hacemos habitualmente con él.

Gestión de Procesos

Proceso

Instancia de ejecución de un programa, con recursos asignados (memoria, ficheros, semáforos, . . .)

El S.O. ofrece servicios de gestión de procesos, que incluyen:

- Creación:
  - A partir de otro proceso, copiando su *mapa de memoria*
- Ejecución de programas:
  - A partir de un ejecutable almacenadas en un fichero en un dispositivo de E/S (disco).
  - El SO carga en memoria las secciones de código y datos e inicializa el estado del procesador para ejecutarlo desde su punto de entrada.
- Planificaión:
  - Reparto de uso de CPU entre procesos
- Comunicación y sincronización:
  - Procesos que ejecutan *concurrentemente* pueden cooperar

Gestión de Memoria

El S.O. ofrece los siguientes servicios:

- Gestión del espacio de direcciones del proceso
  - Añadir y eliminar regiones, mapear ficheros, etc.
- Asignación de memoria *física* a los procesos
  - El SO **reparte** la memoria física entre los procesos
  - Usa la memoria física como una caché de los espacios de direcciones de los procesos
- Protección:
  - El SO mapea los espacios de direcciones en rangos de direcciones físicas que no solapan (Protección entre aplicaciones)
  - Parte del espacio de direcciones sólo accesible en modo kernel (Protección del SO)
  - Diferentes permisos para distintas regiones (seguridad y detección de errores)
- Gestión de la memoria libre, no asignada a ningún proceso

Gestión del sistema de ficheros

El S.O. ofrece:

- Un modelo sencillo de fichero al programador y un API
- Un sistema para agrupar/relacionar ficheros
  - Directorio y API
- Los directorios pueden también contener/agrupar otros directorios
  - Árbol de directorios

y se encarga de forma transparente de:

- Asignar espacio de disco a los ficheros
- Localizar los ficheros en disco a partir de su nombre
- Gestionar espacio libre en disco
- Identificar los ficheros y sus permisos/propietario
- Permitir accesos coordinados desde varios procesos
- Recuperar información ante problemas inesperados (cortes de luz, etc)

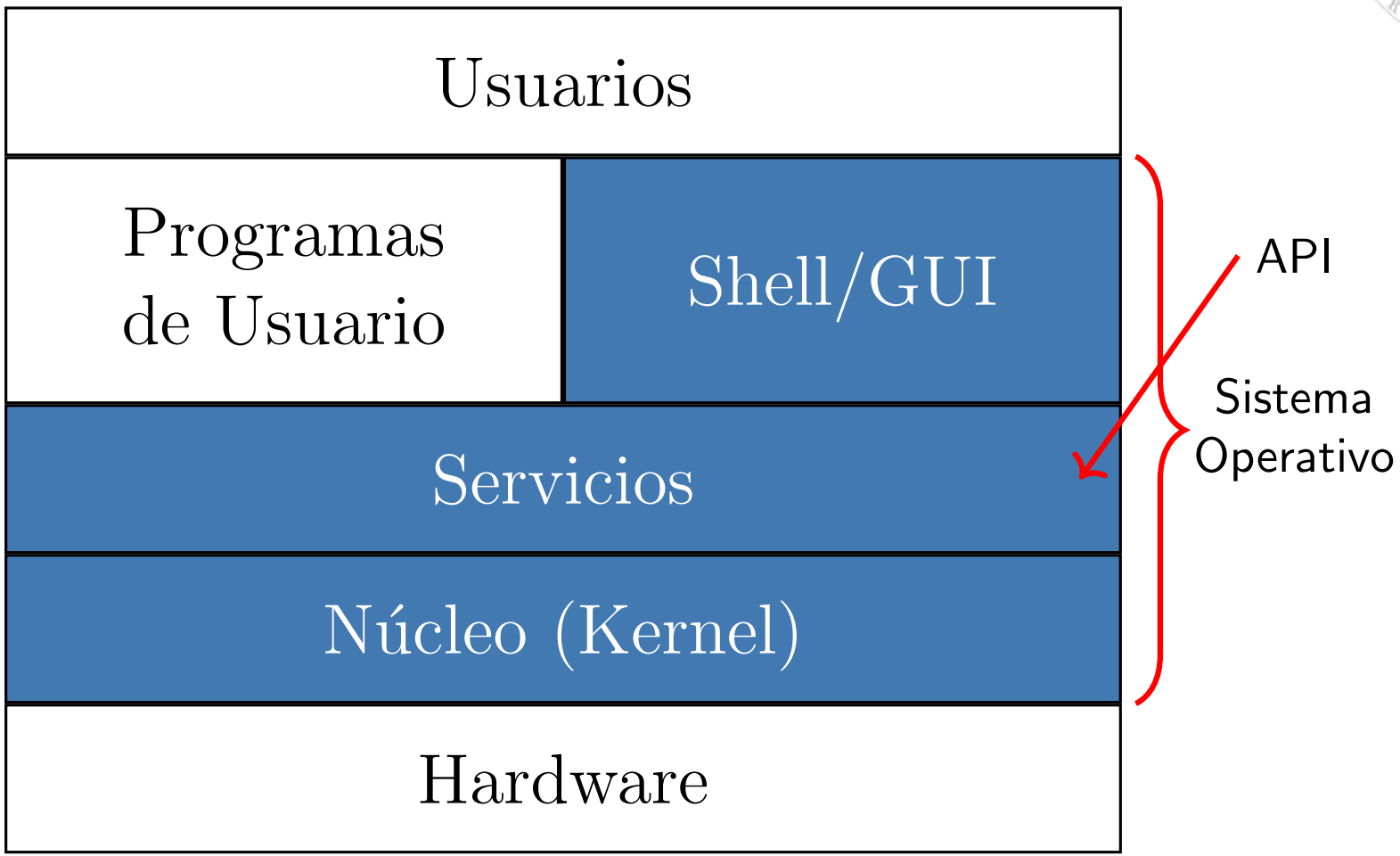
Gestión de la E/S

El S.O. ofrece los siguiente servicios:

- Drivers que se encargan de la gestión de bajo nivel de los dispositivos
  - Recordar el laboratorio de EC
- Un interfaz sencillo para el uso de dispositivos
  - Permite comunicar de forma segura código de aplicación con el driver
- Un mecanismo de carga de drivers
- Facilidades para la implementación de drivers
- Un esquema jerárquico para el diseño
- Un sistema de cache asociado a los dispositivos para mejorar el rendimiento

Agenda

El SO es software



El SO es software

Un programa más al fin y al cabo:

- ¿Cómo puede controlar y restringir el acceso a la memoria y los dispositivos de los demás programas?
- ¿Cómo comienza a ejecutarse ese programa sin los servicios del SO?
- ¿Cómo puede un programa en ejecución obtener los servicios del SO, que en ese momento no se está ejecutando?
- ¿No es demasiada sobrecarga controlar cada acceso a memoria?

Soporte HW

EL SO requiere soporte del HW para realizar sus funciones:

- Distintos modos de ejecución del procesador
- Excepción de Reset para el arranque del sistema
- Interrupciones software, para ofrecer servicios bajo demanda
- *Timer*, que genere interrupciones periódicas, para tareas periódicas del SO
- Soporte HW para la traducción de direcciones y el control de acceso a memoria

Modos de ejecución del procesador

- El procesador ofrece varios modos de ejecución con distintos permisos de acceso a los recursos, como:
  - Subconjunto de instrucciones disponibles
  - Acceso al mapa de E/S
  - Acceso a los registros de soporte de gestión de memoria (MMU)
  - Acceso al registro de estado para la modificación de los Bits de modo
- En GNU/Linux:
  - el núcleo del SO se ejecuta en el 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

Excepción

Evento interno al procesador que desencadena un mecanismo de tratamiento que, básicamente, consiste en:

1. Detener la ejecución de instrucciones (programa actual)
2. Cambiar a *modo privilegiado*
3. Salvar una parte del estado arquitectónico
4. Saltar a una dirección específica: **vector de la excepción**
   - En esa dirección deberá haber una instrucción de salto a la rutina de tratamiento de excepción (RTI o ISR en inglés)
   - El retorno de dicha rutina retomará la ejecución por donde se interrumpió, pasando al mismo tiempo a modo *usuario*

Interrupción

Excepción provocada por un evento externo.

- El tratamiento HW es un poco más complejo
  - El procesador no puede saber cuándo se producirá
- El mecanismo de retorno puede ser también ligeramente distinto

Arranque del SO

Básicamente:

- El HW se inicia con una excepción especial: *Reset*
- El vector de reset está en memoria no volátil (ROM, Flash)
- La RTI de reset está en memoria no volátil (ROM, Flash)
  - El código de la ROM/Flash (BIOS en PC) se encarga de buscar un SO (o gestor de arranque) en los posibles dispositivos de arranque
  - Si lo encuentra copia el primer sector en memoria
  - Salta a la zona de memoria donde lo ha copiado
  - El código copiado se encarga de cargar todo el S.O. en memoria
  - Finalmente comienza la ejecución del S.O.

Acceso a servicios del SO

A través de una *llamada al sistema*:

1. Se invoca una función de biblioteca del sistema, que
   - I. Prepara los argumentos de la llamada al sistema:
     - según la arquitectura por registro o pila
     - uno de los argumentos es el nº de llamada al sistema
   - II. Ejecuta una instrucción especial que provoca una excepción
     - interrupción software, trap en x86, swi en arm
2. La RTI invoca la función del SO que ofrecerá el servicio
   - Se ejecuta en modo privilegiado como la RTI
   - Se obtiene indexando una tabla con el nº de llamada al sistema
   - Tiene acceso a las estructuras del SO
   - No hay cambio de contexto
3. Al finalizar:
   - I. se prepara el valor de retorno
   - II. se realiza el retorno de interrupción retomándose así la ejecución dónde se interrumpió y en modo usuario
   - III. la función de biblioteca retorna y devuelve el resultado al programa

Tareas periódicas y planificador

El SO utiliza un temporizador (*timer*) que genera interrupciones peródicas (*tick*) lo que le permite intervenir para:

- 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

Acceso a memoria y MMU

- En un procesador segmentado, en cada ciclo se pueden producir 1-2 accesos a memoria (más si es superescalar)
- Cada acceso supone:
  - la comprobación de si es una dirección virtual válida
  - la traducción de una dirección virtual a dirección física
  - la comprobación de que el acceso tiene los permisos adecuados
- No es viable que tenga que realizarse todo esto por software
- El SO requiere de una *Memory Management Unit* (MMU) que realice estas tareas en HW (quizá no completas)

Agenda

Estándar POSIX

- POSIX: Portable Operating System Interface for uniX
  - Estándar de sistemas operativos UNIX de IEEE.
  - NO es una implementación. Sólo define una interfaz.
- Objetivo: portabilidad de las aplicaciones entre diferentes plataformas y sistemas UNIX
- Compuesto o dividido en varios 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