Tema 2. Introducción a los sistemas operativos

(PARTE 2: transparencias 28 a 52)

Concepto de hebra

Hasta el momento hemos visto que la tarea fundamental del sistema operativo es gestionar procesos:

- reservarles recursos,
- permitirles intercambiar y compartir información,
- garantizar el uso correcto de los recursos protegiendo su uso,
- sincronizar los recursos

Para ello, el sistema operativo almacena para cada proceso su estado y la propiedad de los recursos.

También hemos visto cómo se realiza este control tanto en monoprocesadores como multiprocesadores:

- monoprocesador multiprogramado: se intercala en el tiempo la ejecución de procesos
- multiprocesador: además es posible que haya múltiples procesos que se ejecuten simultáneamente

Introducimos ahora un nuevo grado de complicación:

- sistemas multihilo: separar el punto de control de la asignación de los recursos (se encargan los procesos tal y como hemos visto) y el de ejecución de los programas (cada proceso se divide en hebras que permite la ejecución del programa de forma independiente al resto de hebras).

La Figura de la transparencia 29 muestra la diferencia entre los hilos y los procesos desde el punto de vista de cómo se gestionan los procesos. A la izquierda, el modelo monohilo (sin hilos):

- el modelo de un proceso incluye el bloque de control, el espacio de direcciones de usuario, pilas de usuario y núcleo para gestionar el paso en la ejecución de unos procesos a otros.
- Mientras se ejecuta un proceso, éste controla los registros del procesador y su contenido se almacena para cuando haya que retomar su ejecución.

En el modelo multihilo:

- Sigue habiendo un único bloque de control del proceso y espacio de direcciones de usuario.
- Ahora hay una pila y un bloque de control por cada hilo, que contiene los valores de los registros, la prioridad de cada hilo y demás información relativa al estado del hilo.

De este modo:

- Todos los hilos de un proceso comparten el estado y los recursos del proceso asociado, residen en el mismo espacio de direcciones de memoria y tienen acceso a los mismos datos: los hilos de un proceso ven cuando alguno de ellos cambia datos en memoria, abre un fichero en modo lectura… → Es necesario planificar una sincronización en las

actividades de los hilos para que no interfieran en el uso incorrecto de los datos (lo veremos más adelante en el curso).

Ventajas de las hebras respecto al modelo de los hilos (hebras):

- Es más rápido crear un hilo para un proceso que un proceso nuevo
- Es más rápido finalizar un hilo que un proceso
- Es más fácil cambiar entre hilos dentro de un mismo proceso que cambiar entre procesos (cambios de contexto más rápidos)
- Los hilos mejoran la comunicación entre los programas: solo la comunicación entre procesos requiere que intervenga el núcleo para proteger los contenidos de los registros y realizar la comunicación
- Aprovechar las técnicas y funciones de la programación concurrente.

Es más eficiente dividir una aplicación en un conjunto de hilos que en un conjunto de procesos independiente

Modelo de estados para las hebras

Al igual que los hilos, las hebras tienen también estados de ejecución para que se puedan sincronizar entre ellas (transparencia 30, modelo de cinco estados):

- Cuando se **crea** un proceso, se **crea** un hilo para dicho proceso
- Posteriormente los hilos pueden crear nuevos hilos para el mismo proceso (cada uno con su propio registro de contexto y espacio de pila
- Cada nuevo hilo, se coloca en la cola de **preparados**
- Cuando un hilo necesita esperar a un evento, se **bloquea** (almacenando los registros de usuario, el contador de programa y los punteros de pila)
- Cuando se produce el evento en espera, el hilo pasa a estado **preparado**
- Cuando se completa un hilo pasa a estado **ejecutado**, se liberan sus registros y pilas

En la transparencia 31 tenéis un ejemplo en el que se intercalan 3 hilos de 2 procesos distintos en un único procesador. La ejecución va pasando de un hilo a otro conforme se producen eventos o termina el tiempo.

<u>Gestión básica de la memoria</u>

En un sistema multiprogramado, la memoria principal disponible suele compartirse entre varios procesos:

- normalmente el programador no sabe a priori donde se almacenarán sus programas en memoria en tiempo de ejecución
- es conveniente poder mover intercambiar los procesos en memoria principal para hacer más eficiente el uso del procesador (especialmente cuando se copia información del disco a memoria y viceversa)
- para iniciar la ejecución de un proceso, el sistema operativo necesita la ubicación de su información de control, de la pila de ejecución y el punto de entrada del proceso
- el hardware del procesador y el sistema operativo se encargan de ir traduciendo de referencias a direcciones relativas de memoria a absolutas.

Carga de procesos

El primer paso en la creación de un proceso es cargar un programa en la memoria principal. El cargador coloca el módulo de carga en la memoria principal comenzando desde la dirección *x*. Se pueden seguir tres técnicas:

- **Carga absoluta**: El módulo de carga dado debe cargarse siempre en la misma ubicación de la memoria principal: se asigna durante la compilación o lo hace directamente el programador (no habitual por las desventajas que supone si hay instrucciones en el programa que suponen alterar las direcciones). Los programas no son reubicables (como veis en la transparencia 34, el programa ejecutable decide ya las direcciones absolutas donde el cargador carga el programa en la memoria principal)
- **Carga reubicable**: El módulo de carga puede ubicarse y ejecutarse en cualquier lugar de la memoria principal: uso de direcciones relativas de memoria por parte del compilador/ensamblador. El comienzo del módulo se carga en la dirección relativa 0 y el resto de direcciones relativas se especifican a partir de esa dirección. Tal y como veis en la transparencia 34, si el cargador carga el programa a partir de la dirección Y en la memoria principal, solo debe añadir Y a las direcciones relativas (diccionario de reubicación proporcionado por el compilador o ensamblador).
    - **Reubicación estática** (transparencia 35): coincide con el proceso explicado en el párrafo anterior (el cargador se encarga de decidir donde se ubica el programa y añade la dirección base a las referencias relativas).
    - **Reubicación dinámica en tiempo real** (transparencia 36): Se postpone el cálculo de la dirección absoluta al tiempo de ejecución. El módulo de carga se carga en la memoria principal con referencias relativas. Hasta que una instrucción no se ejecuta realmente, no se calcula la dirección absoluta. Para ello, se requiere la ayuda del hardware del procesador para no degradar el rendimiento.

Espacio para las direcciones de memoria

- Espacio de direcciones lógico y físico:

Las direcciones que genera un procesador cuando está ejecutando un proceso son lógicas

- Relativas a una dirección 0, igual para todos los procesos,
- Pero los datos se guardan en posiciones físicas de memoria

Hace falta traducir de direcciones lógicas a físicas

- MMU: memory management unit. El hardware necesario para producir esta traducción.

- Mapa de memoria de un ordenador: espacio de memoria direccionable (de acuerdo con el bus de direcciones)
- Mapa de memoria de un proceso: Tamaño total de direcciones lógica del proceso y correspondencia con las direcciones físicas (se almacena en memoria para cada proceso).

Particionamiento/fragmentación de la memoria
La estructura del programa refleja su división lógica, llevándose a cabo una agrupación lógica de la información en bloques de tamaño variable denominados segmentos. Cada uno de ellos tienen información lógica del programa: subrutina, arreglo, etc. Luego, cada espacio de direcciones de programa consiste en una colección de segmentos, que generalmente reflejan la división lógica del programa.

Como veis en la transparencia 39, cuando los procesos asignados han ocupado posiciones no contiguas de memoria dejando demasiados pueden producirse bloques libres de pequeño tamaño en los que no "caben" nuevos procesos. Es un problema similar al que sucede en el disco duro de los ordenadores cuando toca desfragmentarlo.

Paginación y segmentación

Ambas son soluciones al problema de la fragmentación:

- **Paginación**: La memoria principal se divide en porciones más pequeñas de tamaño fijo. Cada proceso también divide su espacio lógico en porciones pequeñas del mismo tamaño fijo (páginas). A cada página se le asigna porciones disponibles en la memoria (marcos de páginas). El tamaño de página es múltiplo de 2 para que el direccionamiento lógico sea transparente al programador/ensamblador/montador y le sea fácil al hardware del procesador realizar la traducción de direcciones.
  Tal y como veis en la transparencia 42, las direcciones lógicas incluyen un número de página y el desplazamiento dentro de la página. Las direcciones físicas se dividen en número de marco y desplazamiento.

  ¿Qué sucede si no hay marcos consecutivos suficientes para almacenar un proceso?
  El sistema operativo mantiene una **tabla de páginas** por cada proceso, que almacena la ubicación del marco por cada página del proceso (número de marco + modo de acceso autorizado).
  La traducción de direcciones lógicas a físicas las realiza el hardware del procesador: la tabla de marcos de página usada por el sistema operativo contiene la información sobre cada marco de página. En la transparencia 45 tenéis un ejemplo del esquema que se sigue para la traducción, con su correspondiente explicación en la transparencia 46:
    - La tabla de páginas se almacena en la memoria principal.
    - Se utiliza un registro (RBTP, registro base de la tabla de páginas) para almacenar un puntero a la tabla de páginas

  Problema: Necesario dos accesos a memoria para acceder a cada dato/instrucción: 1) Acceder a la tabla de páginas para buscar la entrada 2) Acceder a la memoria para buscar el dato.

  Solución (Transparencias 47 y 48): Utilizar una caché especial de lectura/escritura de alta velocidad denominada buffer de traducción adelantada (TLB): Se organiza en un conjunto de registros que contienen las entradas de la

tabla de páginas que se han utilizado más recientemente. Se consulta primero la TLB para buscar si la entrada de la tabla de páginas solicitada está presente (acierto: se recupera el número de marco y se construye la dirección física) o no (se busca la entrada en la tabla de páginas y se actualiza la TLB con esta nueva entrada).

- **Segmentación**: Al igual que en la paginación, se trocea el espacio lógico en unidades más pequeñas, pero de longitud variable. Cada proceso se divide en segmentos (no necesariamente de la misma longitud). Un proceso se carga cargando todos sus segmentos en particiones dinámicas no necesariamente contiguas (transparencia 49).
  Al igual que la paginación, una dirección lógica está compuesta de un número de segmento y un desplazamiento.
  La segmentación es normalmente visible al programador. Se utiliza una tabla de segmentos por cada proceso, donde cada entrada incluye (entre otras cosas) la base (dirección física donde reside el segmento en memoria), pero además ahora también la longitud del segmento.
  De forma similar a la paginación, la implementación de la segmentación:
  - La tabla de segmentos se mantiene en la memoria principal,
  - Se utiliza un registro que apunta a la tabla de procesos (registro base de la tabla de segmentos, RBTS).
  - Se utiliza un segundo registro que almacena el número de segmentos del proceso (registro longitud de la tabla de segmentos, STLR).
  - Tal y como tenéis en la transparencia 52, se realizan dos comprobaciones: 1) que el desplazamiento en la dirección virtual sea menor que el tamaño (registro STLR), 2) que el acceso esté autorizado de acuerdo con los permisos de protección en la tabla de segmentos.