Materia: Sistemas Operativos
Unidad: Hilos (threads)
Versión: Abril 2005
Palabras claves: hilos, threads, ULT(user level thread), KLT (kernel level thread), SO (sistemas operativos), RR (round robin), PCB (process control block), TCB (thread control block), API (aplication program interface), LWP (Light Weight Process)

Hilos (Threads)

Conceptos Introductorios

Los nuevos sistemas operativos (por ejemplo W2000, Solaris, Linux) cuentan con un mecanismo para proveer concurrencia dentro del mismo espacio de direcciones. A este tipo de "proceso" se le llama ***hilo (thread)***. También se le llama *lightweight process (LWP)*.

Supongamos un ambiente de servidor de archivos. Cada vez que llega una solicitud para acceder a un nuevo archivo, se debe crear una unidad de atención. Pero esta unidad de atención tendrá un ciclo de vida corto y, además, puede haber una gran demanda de estas unidades que se creerán y destruirán en un corto período.
Si es un ambiente multiprocesador estas unidades pueden ejecutarse simultáneamente en los diferentes procesadores. Que estas unidades de atención tengan estructura de hilos mejorará la productividad del sistema.

Para ejemplificar, en el modelo productor-consumidor se comparte un buffer donde el productor almacena lo producido y el consumidor lo usa. Es un espacio compartido. Podríamos implementar este modelo como una tarea con dos hilos (el productor y el consumidor). De esa manera, el switching entre el hilo productor y el hilo consumidor se hace dentro del mismo espacio de direcciones, siendo el buffer compartido parte de ese espacio, y según la implementación, puede ser sin intervención del SO.

Otros casos donde es beneficioso el uso de hilos es en las aplicaciones de procesos de comunicaciones y control de transacciones.

Ya vimos que cuando un proceso tradicional crea a otro del mismo tipo, se genera un proceso hijo con otro PC (program counter) y un nuevo espacio de direcciones. En cambio, un hilo creado por un proceso tendrá su propio contexto, pero se ejecutará dentro del mismo espacio de direcciones que el proceso que lo creó. No sólo compartirá recursos sino que se ejecutará en el mismo espacio de direcciones.

El hilo es la unidad básica de utilización de la CPU.

Para establecer una diferencia entre el concepto de proceso y de hilo, debemos distinguir entre la unidad de propiedad de recursos (el proceso) y la unidad de ejecución (el hilo).

Un hilo es la unidad de trabajo que se puede someter a ejecución. Se ejecuta secuencialmente y es interrumpible para que el procesador pase a otro hilo. Permite que el programador pueda ejercer la modularidad.
Un proceso puede verse como una colección de uno o más hilos.

Cuando un sistema operativo soporta varios hilos en ejecución dentro del mismo proceso se dice que es *multihilo*.

Uso de hilos en un sistema multitarea

Veamos cuatro situaciones donde usar hilos en un ambiente multitarea.

Para trabajo interactivo, mejorando el grado de respuesta: La mayoría del software de escritorio que utilizamos hoy, trabajan con hilos. Por ejemplo, un navegador de red tiene un hilo que va mostrando las imágenes y otro recupera datos. En un procesador de texto, un hilo podría ir manejando la edición del texto, mientras otro analiza la ortografía. O en una planilla de calculo, un hilo lee las entradas del usuario mientras otro ejecuta las ordenes y actualiza.
Como consecuencia se concibe un aumento de la velocidad en la ejecución de la aplicación.
También pensemos en un servidor de red que acepta solicitudes de páginas WEB por parte de los clientes. SI este servidor se ejecutara como un proceso tradicional de un solo hilo, atendería a un cliente por vez. También podría haber un proceso receptor de solicitudes que por cada una que llegue cree un proceso hijo para que la atienda, con toda la actividad (y gasto de recursos) que significa la creación de nuevos procesos. Se obtendrá una mayor performance si tengo un proceso que atiende que crea hilos en su espacio.

Para proceso asíncrono: Puede diseñarse un sistema con hilos para que asincrónicamente la información sea llevada de la RAM al disco previniendo los problemas por cortes de alimentación, o para backups periódicos.

Para aceleración de la ejecución: Para lectura de datos, un hilo puede procesar un lote de datos, mientras otro hilo lee el lote siguiente.

Para organización de los programas, economizando recursos: Los programas que tienen gran actividad, con entradas y salidas con diferentes orígenes y destinos, pueden diseñarse e implementarse con hilo para su mejor organización. Además, es mucho menos costoso a nivel de consumo de recursos pues los hilos los comparten y al estar en el mismo especio de direcciones, las operaciones entre hilos son mucho más ágiles que las operaciones entre procesos.

Ejemplo: Hilos en Java

Java no tiene soporte para eventos asíncronos. Si un proceso Java trata de hacer una conexión con un servidor, el cliente se bloquea hasta que se hace la conexión o hay un timeout.
Para manejar esto, un hilo hace la conexión con el servidor, y otro hilo, inicialmente dormido, se despertará luego que transcurra el tiempo estimado de timeout. Este hilo, al que llamaremos temporizador, analiza si el hilo de conexión está aun tratando de conectarse al servidor. SI aun no lo hizo, el hilo temporizador activa una interrupción y controlará que ningún otro trate de conectarse.

Diferencias con el concepto tradicional

Estructura

En un ambiente multihilo, un proceso es la unidad de protección y asignación de recursos.
Así, cada hilo dentro de un proceso contará con un estado de ejecución, un contexto de procesador, una pila en modo usuario y otra en modo supervisor, almacenamiento para variables locales y acceso a memoria y recursos del proceso (archivos abiertos, señales, además de la parte de código y datos) que compartirá con el resto de los hilos.
Todos juntos, los hilos de un proceso constituyen una tarea.

La estructura de un hilo está constituida por:

- un program counter
- un conjunto de registros
- un espacio de stack

Un proceso tradicional (*heavyweight process o monohilo*) es una tarea formada por un solo hilo. Cuenta con un PCB (process control block), espacio de direcciones de usuario y 2 pilas.

En un proceso multihilo, además de PCB, espacio de direcciones de usuario y 2 pilas, cada hilo tiene su propio TCB (thread control block) y 2 pilas.

Los hilos de un proceso comparten estado y recursos del sistema, residiendo en el mismo espacio de direcciones y accediendo a los mismos datos.

Context switch

Cuando un proceso tradicionales gana CPU, el sistema operativo debe hacer un cambio de contexto (context switch) para hacer el correspondiente salvado del ambiente relacionado con el proceso anterior y cargar el ambiente del nuevo. Esta actividad la realiza el sistema operativo, en modo supervisor, y es la consecuencia de algún system call emitido por el proceso saliente o interrupción.

Entre los hilos que comparten una tarea, el cambio entre uno y otro es más simple pues el context switch es sólo a nivel del conjunto de registros y carga no hay cambios con respecto a espacios de direcciones.
Podemos resumir que en el switching de los hilos no se exige la participación del sistema operativo e interrupciones al kernel.

La creación de un hilo

La creación de un proceso involucra una operación tipo fork, creando un nuevo espacio de direcciones, PCB, PC, etc. El nuevo proceso es creado y controlado a través de system calls que exigen el mismo tratamiento que una interrupción.
La creación de un hilo involucra sólo la creación de la estructura detallada más arriba: un conjunto de registros, un PC y un espacio para stack, requiriendo un gasto mínimo de procesamiento.

Operación

Los hilos pueden ser una manera eficiente de permitir que un servidor pueda atender varios requerimientos.
En un ambiente monoprocesador, un hilo comparte la CPU con los otros hilos, y sólo uno está activo en un momento dado. Se ejecuta secuencialmente y tiene su propio PC y stack.
Puede tener hilos hijos y bloquearse a la espera de un system call. Si se bloquea un hilo, puede ejecutarse otro hilo.
Si se hace swapping del proceso, será acompañado por sus hilos. Al terminar un proceso, terminan todos sus hilos.

Estados de un hilo

Un hilo puede estar en estado de ready, blocked, running ó exiting, como los procesos tradicionales.
Hay cuatro operaciones que provocan los cambios de estado: la creación, el bloqueo, el desbloqueo y la terminación.

Al crearse un proceso, se crea un hilo (que puede crear otros hilos, cada uno con su nuevo contexto y pilas) y pasará al estado de listo.

Cuando el hilo debe esperar por un evento se bloquea. Según las diferentes implementaciones, puede ocurrir que quede todo el proceso bloqueado, o solo ese hilo. En este último caso se le puede dar el control a otro hilo de ese proceso.

Cuando se produce el evento por el que estaba esperando, el hilo bloqueado se desbloquea, pasando al estado de listo.

Al terminar, el hilo libera su contexto y sus pilas.

Beneficios del uso de los hilos

- Es más rápido crear un nuevo hilo que un nuevo proceso.
- Un hilo, al terminar, sólo libera el espacio asignado, siendo este procedimiento mucho más rápido que la terminación de un proceso.

- Los hilos, además, mejoran la comunicación entre procesos. Mientras los procesos tradicionales necesitan la intervención del núcleo del SO para protección y administración del mecanismo de comunicación, la comunicación entre hilos del mismo proceso se puede hacer sin la intervención del núcleo, pues comparten memoria y archivos.

Tipos de hilos

Hay **hilos a nivel de usuario** (ULT, user level thread) e **hilos a nivel de núcleo** (KLT, kernel level thread).

En los ULT, la administración de los hilos lo hace la aplicación sin intervención del kernel. Es más: el kernel ni se entera de su existencia. El kernel no ve el hilo: ve un proceso haciendo un requerimiento. LA creación de los hilos y operaciones se hacen en a nivel de usuario y por lo tanto son más rápidos de crear y utilizar.

En estos casos se trabaja con una biblioteca de hilos que son funciones para implementar ULT invocadas desde la aplicación (crear y destruir hilos, intercambio de mensajes y datos entre hilos, planificación de ejecución, salvado y restauración de contexto).
Todo esto se realiza dentro del mismo proceso, en su espacio de direcciones.

Ejemplos de bibliotecas de Hilos son POSIX Pthreads, Mach C-Threads y Solaris Threads.

Las ventajas de los ULT son:

- No interviene el núcleo en el intercambio de hilos (las estructuras de datos están todas dentro del espacio de direcciones del proceso).
- Se puede planificar la ejecución de los hilos dentro de cada proceso como se prefiera (prioridades, RR). O sea: cada proceso administra sus hilos como le convenga al programador, aunque use una planificación distinta de la que usa el kernel para los procesos.
- Los ULT se pueden ejecutar en cualquier SO pues no dependen de él.

Los ULT tienen también, desventajas:

- Los hilos no son independientes entre ellos, pues pueden acceder a cualquier dirección dentro de la tarea, y un hilo puede leer o escribir en el stack de otro. ***No hay protección entre hilos***.
- La suspensión de un hilo, como puede ser por una llamada al sistema, bloquea a todo el proceso, o sea, todos los hilos de ese proceso.
- En un ambiente multiprocesador, cada procesador ejecuta un proceso. Por lo tanto no voy a poder aprovechar la potencialidad de estos ambientes y los hilos de un proceso usarán un único procesador (no se puede implementar una concurrencia entre los hilos de un mismo proceso ejecutándose paralelamente entre los distintos procesadores).

En los KLT, el trabajo de gestión de hilos lo hace el núcleo. La aplicación gestiona el hilo a través de una API. Existen un conjunto de system calls similar a la de los procesos específicas para hilos). Esta metodología se usa en W2000, Linux, OS/2.

El kernel mantiene información del proceso en general y de cada hilo del proceso en particular. La planificación la hace el kernel en base a los hilos.

La ventaja es que en un ambiente multiprocesador, los hilos de un mismo proceso pueden estar ejecutándose concurrentemente en los distintos procesadores (la unidad de asignación del procesador ya no es el proceso).

La desventaja de los KLT es que el pase del control entre hilos de un mismo proceso necesita un cambio de modo. Esto hace prever que la creación y administración de los KLTs es más lenta que los ULTs.

Modelos multihilos

Como algunos sistemas proveen tanto KLT como ULT, tenemos diferentes modelos multihilo: varios a uno, uno a uno y varios a varios

Modelo varios a uno

Este modelo relaciona varios hilos a nivel de usuario con un hilo de kernel.
Si bien toda la operación entre hilos se hace en el espacio de usuario, si uno de ellos se bloquea, se bloquea todo el proceso. Por ejemplo, ante un system call que realice un hilo, se bloqueará todo el proceso.
Al kernel acceden de a un hilo, asi que este modelo no es util en ambiente multiprocesador.

Modelo uno a uno

En este modelo cada hilo de usuario se relaciona con uno de kernel.
En este caso, si un hilo se bloquea, puede ejecutarse otro hilo del mismo proceso. Lo que si garantiza este modelo es la concurrencia pues en un proceso formado por varios hilos, en un ambiente multiprocesador puede correr cada uno en un procesador distinto.
No obstante se debe tener en cuenta que cada vez que necesito un hilo de usuario se debe crear uno de kernel con el gasto que eso significa.

Modelo de varios a varios

Este modelo combina o “multiplexa” muchos hilos a nivel de usuario con un número menor o igual de hilos a nivel de kernel.
Se pueden crear tantos hilos de usuario como sea necesario y los hilos a nivel de kernel correspondientes se ejecutan en paraleo si es un ambiente multiprocesador.
En el modelo uno a uno si bien premite mayor concurrencia, hay que ser cuidadoso con no crear demasiados hilos, pues en algunos casos este número se limita.
Si es un modelo varios a varios se crearan un número de hilos kernel que se permita y se multiplexan los hilos de usuario en esos hilos de kernel.

Hilos en Solaris

Solaris es una versión de Unix. Es un sistema operativo moderno que soporta threads a nivel de usuario y de kernel, SMP, planificación en tiempo real.

Los ULT son soportados por una librería y el kernel no los reconoce.

Entre los hilos a nivel de usuario y los a nivel de kernel, hay un nivel intermedio, LWP (Light Weight Process).
Cada proceso tiene como mínimo un LWP. La biblioteca de hilos se encarga de combinar hilos ULT con respecto a un LWP.

Los ULT se multiplexan en los LWPs del proceso y solamente trabajan los ULT del LWP corriente. El resto espera por un LWP que se ejecute o están bloqueados.

Las actividades dentro del kernel las deben ejecutar los KLT. Hay un KLT por cada LWP y hay KLT que se ejecutan dentro del kernel que no tienen LWP asociado.

El kernel aplica el scheduler a los KLT (por procesador; si hay SMP, puede distribuirlos). Puedo asignar un thread a una CPU, para que se ejecute sólo en ella o el procesador sólo dedicado a esa tarea.

Los ULT se schedulean dentro de los LWP sin que intervenga el kernel (no es necesario el context switch). Lo que los hace muy eficientes.
ULTs son soportados por los LWP.

Cada LWP está conectado a un KLT. Puede haber varios LWP en una tarea que se usan cuando los threads necesitan trabajar con el kernel.

Hay ULTs ligados y no ligados. Por default, son no ligados.
Los ligados están unidos permanentemente a un LWP. Es útil en procesos que exigen un tiempo de respuesta corto, por ejemplo en tiempo real.

El ULT no ligado no está unido permanentemente a un LWP. Los no ligados se multiplexan con los LWPs disponibles.

Entonces: un proceso puede tener varios ULTs que la biblioteca de hilos multiplexa con los LWPs disponibles, sin intervención del kernel. Cada LWP está conectado con un KLT. Si bien en un proceso puede haber varios LWP sólo se usan cuando el hilo necesita comunicarse con el kernel.

Supongamos que se quiere trabajar con 5 archivos y hay 5 read que deben ocurrir simultáneamente.
Debe haber 5 LWP donde cada ULT haga el request para que se canalicen como requerimiento de I/O simultáneos que puedan llevarse a distintas CPUs (lo ideal se fueran distintos discos...).
Si fueran 4 LWP, el 5to debería esperar el retorno de uno de los LWP para ejecutarse.

Lo que se schedulea a nivel del kernel son los KLT (en 1 o más CPUs). Si se bloquea un KLT el procesador puede correr otro.
En un KLT bloqueado, se bloquea su LWP. Si un proceso tiene más de un KLT puede continuar con los no bloqueados.
La biblioteca de hilos realiza además estas actividades:

- Ajusta el número de LWPs para mejorar el rendimiento de la aplicación. Si todos los LWPs de un proceso están bloqueados y hay más ULTs, la biblioteca crea otro LWP para asignar a un ULT que espera. Pero no olvidemos que se crea un hilo de kernel adicional
- elimina los LWP cuando no se utilizan por cierto tiempo.

KLT tiene una pequeña estructura de datos, y un stack. Esta estructura incluye una copia de registros del kernel, apuntador al LWP al que está relacionado, información de prioridades y planificación.

LWP tiene registros para el ULT con el que está ligado, información de memoria y de administración. Un LWP es una estructura de datos del kernel y reside en el espacio de kernel.

ULT tiene un identificador de hilo, un conjunto de registros (incluye PC y apuntador de pila), la pila y su prioridad. Esta prioridad la usa la biblioteca de hilos para planificación. Nada de esto es recurso de kernel y todo está en el espacio de usuario. EL kernel no participa en el scheduling de los ULTs.

Aunque haya cientos de ULTs, el kernel sólo vé LWPs.

En Solaris 2 , la PCB de un proceso tendrá, entre otros datos, la identificación del proceso, un mapa de memoria, la lista de archivos abiertos, la prioridad y un apuntador a una lista de LWPs asociados con el proceso.