Hilo de ejecución

Este artículo o sección necesita **referencias** que aparezcan en una **publicación acreditada**, como revistas especializadas, monografías, prensa diaria o páginas de Internet fidedignas.
Puedes añadirlas **así** o avisar al autor principal del artículo [1] en su página de discusión pegando: {{subst:Aviso referencias|Hilo de ejecución}} ~~~~

En sistemas operativos, un **hilo de ejecución** o subproceso es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.

Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).

El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

Algunos lenguajes de programación tienen características de diseño expresamente creadas para permitir a los programadores lidiar con hilos de ejecución (como Java o Delphi). Otros (la mayoría) desconocen la existencia de hilos de ejecución y éstos deben ser creados mediante llamadas de biblioteca especiales que dependen del sistema operativo en el que estos lenguajes están siendo utilizados (como es el caso del C y del C++).

Un ejemplo de la utilización de hilos es tener un hilo atento a la interfaz gráfica (iconos, botones, ventanas), mientras otro hilo hace una larga operación internamente. De esta manera el programa responde de manera más ágil a la interacción con el usuario. También pueden ser utilizados por una aplicación servidora para dar servicio a múltiples clientes.

Diferencias entre hilos y procesos

Los hilos se distinguen de los tradicionales procesos en que los procesos son −generalmente− independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, muchos hilos generalmente comparten otros recursos de forma directa. En muchos de los sistemas operativos que dan facilidades a los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen. Al cambiar de un proceso a otro el sistema operativo (mediante el *dispatcher*) genera lo que se conoce como *overhead*, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (*context switch*), en este caso pasar del estado de ejecución (*running*) al estado de espera (*waiting*) y colocar el nuevo proceso en ejecución. En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.

Sistemas operativos como Windows NT, OS/2 y Linux (2.5 o superiores) dicen tener hilos "baratos", y procesos "costosos" mientras que en otros sistemas no hay una gran diferencia.

Funcionalidad de los hilos

Al igual que los procesos, los hilos poseen un estado de ejecución y pueden sincronizarse entre ellos para evitar problemas de compartimiento de recursos. Generalmente, cada hilo tiene una tarea especifica y determinada, como forma de aumentar la eficiencia del uso del procesador.

Estados de un hilo

Los principales estados de los hilos son: Ejecución, Listo y Bloqueado. No tiene sentido asociar estados de suspensión de hilos ya que es un concepto de proceso. En todo caso, si un proceso está expulsado de la memoria principal (ram), todos sus hilos deberán estarlo ya que todos comparten el espacio de direcciones del proceso.

Cambio de estados

- *Creación:* Cuando se crea un proceso se crea un hilo para ese proceso. Luego, este hilo puede crear otros hilos dentro del mismo proceso, proporcionando un puntero de instrucción y los argumentos del nuevo hilo. El hilo tendrá su propio contexto y su propio espacio de la columna, y pasara a la final de los listos.
- *Bloqueo:* Cuando un hilo necesita esperar por un suceso, se bloquea (salvando sus registros de usuario, contador de programa y punteros de pila). Ahora el procesador podrá pasar a ejecutar otro hilo que esté en la final de los Listos mientras el anterior permanece bloqueado.
- *Desbloqueo:* Cuando el suceso por el que el hilo se bloqueó se produce, el mismo pasa a la final de los Listos.
- *Terminación:* Cuando un hilo finaliza se liberan tanto su contexto como sus columnas..

Ventajas de los hilos contra procesos

Si bien los hilos son generados a partir de la creación de un proceso, podemos decir que un proceso es un hilo de ejecución, conocido como Monohilo. Pero las ventajas de los hilos se dan cuando hablamos de Multihilos, que es cuando un proceso tiene múltiples hilos de ejecución los cuales realizan actividades distintas, que pueden o no ser cooperativas entre sí. Los beneficios de los hilos se derivan de las implicaciones de rendimiento.

1. Se tarda mucho menos tiempo en crear un hilo nuevo en un proceso existente que en crear un proceso. Algunas investigaciones llevan al resultado que esto es así en un factor de 10.
2. Se tarda mucho menos en terminar un hilo que un proceso, ya que cuando se elimina un proceso se debe eliminar el BCP del mismo, mientras que un hilo se elimina su contexto y pila.
3. Se tarda mucho menos tiempo en cambiar entre dos hilos de un mismo proceso
4. Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución. En la mayoría de los sistemas en la comunicación entre procesos debe intervenir el núcleo para ofrecer protección de los recursos y realizar la comunicación misma. En cambio, entre hilos pueden comunicarse entre sí sin la invocación al núcleo. Por lo tanto, si hay una aplicación que debe implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de hilos que con una colección de procesos separados.

Sincronización de hilos

Todos los hilos comparten el mismo espacio de direcciones y otros recursos como pueden ser archivos abiertos. Cualquier modificación de un recurso desde un hilo afecta al entorno del resto de los hilos del mismo proceso.Por lo tanto, es necesario sincronizar la actividad de los distintos hilos para que no interfieran unos con otros o corrompan estructuras de datos.

Una ventaja de la programación multihilo es que los programas operan con mayor velocidad en sistemas de computadores con múltiples CPUs (sistemas multiprocesador o a través de grupo de máquinas) ya que los hilos del programa se prestan verdaderamente para la ejecución concurrente. En tal caso el programador necesita ser cuidadoso para evitar condiciones de carrera (problema que sucede cuando diferentes hilos o procesos alteran datos que otros también están usando), y otros comportamientos no intuitivos. Los hilos generalmente requieren reunirse para procesar los datos en el orden correcto. Es posible que los hilos requieran de operaciones atómicas para impedir que los datos comunes sean cambiados o leídos mientras estén siendo modificados, para lo que usualmente se utilizan los semáforos. El descuido de esto puede generar interbloqueo.

Formas de multihilos

Los sistemas operativos generalmente implementan hilos de dos maneras:

- *Multihilo apropiativo:* permite al sistema operativo determinar cuándo debe haber un cambio de contexto. La desventaja de esto es que el sistema puede hacer un cambio de contexto en un momento inadecuado, causando un fenómeno conocido como inversión de prioridades y otros problemas.
- *Multihilo cooperativo:* depende del mismo hilo abandonar el control cuando llega a un punto de detención, lo cual puede traer problemas cuando el hilo espera la disponibilidad de un recurso.

El soporte de *hardware* para multihilo se encuentra disponible desde hace relativamente poco tiempo. Esta característica fue introducida por Intel en el Pentium 4, bajo el nombre de HyperThreading.

Usos más comunes

Los usos más comunes son en tecnologías SMPP y SMS para la telecomunicaciones aquí hay muchísimos procesos corriendo a la vez y todos requiriendo de un servicio.

Trabajo interactivo y en segundo plano

Por ejemplo, en un programa de hoja de cálculo un hilo puede estar visualizando los menús y leer la entrada del usuario mientras que otro hilo ejecuta las órdenes y actualiza la hoja de calculo. Esta medida suele aumentar la velocidad que se percibe en la aplicación, permitiendo que el programa pida la orden siguiente antes de terminar la anterior.

Procesamiento asíncrono

Los elementos asíncronos de un programa se pueden implementar como hilos. Un ejemplo es como los softwares de procesamiento de texto guardan archivos temporales cuando se está trabajando en dicho programa. Se crea un hilo que tiene como función guardar una copia de respaldo mientras se continúa con la operación de escritura por el usuario sin interferir en la misma.

Aceleración de la ejecución

Se pueden ejecutar, por ejemplo, un lote mientras otro hilo lee el lote siguiente de un dispositivo.

Estructuración modular de los programas

Puede ser un mecanismo eficiente para un programa que ejecuta una gran variedad de actividades, teniendo las mismas bien separadas mediante hilos que realizan cada una de ellas.

Implementaciones

Hay dos grandes categorías en la implementación de hilos:

- Hilos a nivel de usuario.
- Hilos a nivel de kernel.

También conocidos como **ULT** (*user level thread*) y **KLT** (*kernel level thread*)

Hilos a nivel de usuario (ULT)

En una aplicación ULT pura, todo el trabajo de gestión de hilos lo realiza la aplicación y el núcleo o kernel no es consciente de la existencia de hilos. Es posible programar una aplicación como multihilo mediante una biblioteca de hilos. La misma contiene el código para crear y destruir hilos, intercambiar mensajes y datos entre hilos, para planificar la ejecución de hilos y para salvar y restaurar el contexto de los hilos.

Todas las operaciones descritas se llevan a cabo en el espacio de usuario de un mismo proceso. El kernel continua planificando el proceso como una unidad y asignándole un único estado (Listo, bloqueado, etc.).

Ventajas de los ULT

- El intercambio de los hilos no necesita los privilegios del modo kernel, porque todas las estructuras de datos están en el espacio de direcciones de usuario de un mismo proceso. Por lo tanto, el proceso no debe cambiar a modo kernel para gestionar hilos. Se evita la sobrecarga de cambio de modo y con esto el sobrecoste u overhead.
- Se puede realizar una planificación específica. Dependiendo de que aplicación sea, se puede decidir por una u otra planificación según sus ventajas.
- Los ULT pueden ejecutar en cualquier sistema operativo. La biblioteca de hilos es un conjunto compartido.

Desventajas de los ULT

- En la mayoría de los sistemas operativos las llamadas al sistema (System calls) son bloqueantes. Cuando un hilo realiza una llamada al sistema, se bloquea el mismo y también el resto de los hilos del proceso.
- En una estrategia ULT pura, una aplicación multihilo no puede aprovechar las ventajas de los multiprocesadores. El núcleo asigna un solo proceso a un solo procesador, ya que como el núcleo no interviene, ve al conjunto de hilos como un solo proceso.

Una solución al bloqueo mediante a llamadas al sistema es usando la técnica de jacketing, que es convertir una llamada bloqueante en no bloqueante.

Hilos a nivel de núcleo (KLT)

En una aplicación KLT pura, todo el trabajo de gestión de hilos lo realiza el kernel. En el área de la aplicación no hay código de gestión de hilos, únicamente un API (interfaz de programas de aplicación) para la gestión de hilos en el núcleo. Windows 2000, Linux y OS/2 utilizan este método. Linux utiliza un método muy particular en que no hace diferencia entre procesos e hilos, para linux si varios procesos creados con la llamada al sistema "clone" comparten el mismo espacio de direcciones virtuales el sistema operativo los trata como hilos y lógicamente son manejados por el kernel.

Ventajas de los KLT

- El kernel puede planificar simultáneamente múltiples hilos del mismo proceso en múltiples procesadores.
- Si se bloquea un hilo, puede planificar otro del mismo proceso.
- Las propias funciones del kernel pueden ser multihilo

Desventajas de los KLT

- El paso de control de un hilo a otro precisa de un cambio de modo..

Combinaciones ULT y KLT

Algunos sistemas operativos ofrecen la combinación de ULT y KLT, como Solaris.

La creación de hilos, así como la mayor parte de la planificación y sincronización de los hilos de una aplicación se realiza por completo en el espacio de usuario. Los múltiples ULT de una sola aplicación se asocian con varios KLT. El programador puede ajustar el número de KLT para cada aplicación y máquina para obtener el mejor resultado global.

En un método combinado, los múltiples hilos de una aplicación se pueden ejecutar en paralelo en múltiples procesadores y las llamadas al sistema bloqueadoras no necesitan bloquear todo el proceso.

Véase también

- Planificador
- POSIX

Referencias

[1] http://en.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n

Fuentes y contribuyentes del artículo

Hilo de ejecución *Fuente*: http://es.wikipedia.org/w/index.php?oldid=39832697 *Contribuyentes*: Alexman321, Antipatico, Balderai, Beto29, Biasoli, Camilo, CayoMarcio, Ciencia Al Poder, Cratón, Diegusjaimes, Dodo, Dogor, Edub, GermanX, Gothmog, Jachguate, Jag2k4, Jarke, Javier Losada, Javierito92, Julie, Kur4i, Leon95, Lev, Martincarr, Matdrodes, Mcongom, Murphy era un optimista, Niqueco, Nubecosmica, PabloCastellano, PoLuX124, Roberto Parrillas, Super braulio, Taichi, Tomatejc, Yrithinnd, Yucon, 151 ediciones anónimas

Fuentes de imagen, Licencias y contribuyentes

Imagen:Question book.svg *Fuente*: http://es.wikipedia.org/w/index.php?title=Archivo:Question_book.svg *Licencia*: GNU Free Documentation License *Contribuyentes*: Diego Grez, Javierme, Loyna, Remember the dot, Victormoz, 4 ediciones anónimas

Licencia