UNIVERSIDAD DE ANTIOQUIA

1 8 0 3

Facultad de Ingeniería

Departamento de electrónica

Curso de informática II

HILOS
PROYECTO DE INVESTIGACION

por:

Sebastian Bonilla Cruz

Medellín, Antioquia, Colombia
Julio, 2020

¿Qué son los Hilos?

Cada microprocesador se compone fundamentalmente de 2 partes primordiales: Los nucleos/cores y los hilos/threads. Estos núcleos son, cada uno de ellos, y explicado de la forma más sencilla posible, una unidad de procesamiento en sí misma. Los núcleos se dedican a leer instrucciones y a ejecutar acciones específicas. Por otro lado, los hilos, son un elemento muy alejado de lo que es un núcleo. Este elemento se encarga del flujo de control de programa y su tarea directa es ayudar al procesador en la administración de sus tareas. El núcleo ejecuta las tareas y los hilos las administran. Gracias a los hilos, las unidades mínimas de asignación, que son las tareas o procesos de un programa, pueden dividirse en trozos para así optimizar los tiempos de espera de cada instrucción en la cola del proceso. Estos trozos se llaman subprocesos o threads. Dicho de otra forma, cada hilo de procesamiento contiene un trozo de la tarea a realizar, algo más simple de realizar que si introducimos la tarea completa en el núcleo físico. De esta forma la CPU es capaz de procesar varias tareas al mismo tiempo y de forma simultánea. De hecho, podrá hacer tantas tareas como hilos tenga; y normalmente son una o dos por cada núcleo. Por todo lo anterior se puede concluir que los hilos son una funcionalida que controla el flujo de control de datos de un programa.

1. HISTORIA

La noción de un hilo, como un flujo secuencial de control, se remonta a 1965, al menos, con el sistema Berkeley Timesharing. Solo que no se llamaban hilos en ese momento, sino procesos [Dijkstra, 1965]. Los procesos interactúan a través de variables compartidas, semáforos y medios similares. Max Smith realizó una implementación de prototipos de hilos en Multics (Sistema Operativo de tiempo compartido) alrededor de 1970; utilizó múltiples pilas en un solo proceso pesado para soportar compilaciones de fondo.
Quizás el progenitor más importante de subprocesos es el lenguaje de programación PL / 1, de aproximadamente el período de 1965. El lenguaje definido por IBM proporcionó una TAREA CALL XXX (A, B); construir, que bifurcó un hilo para XXX. No está claro si algún compilador de IBM implementó esta característica, pero se examinó de cerca mientras se diseñaba Multics, se decidió que la llamada TASK como se definió no se asignaba a los procesos, ya que no había protección entre los hilos de control. Por lo tanto, Multics tomó una dirección diferente, y la función TASK fue eliminada de PL / 1 por IBM en cualquier caso, junto con el atributo ABNORMAL y muchas otras cosas innecesarias.
Luego vino Unix, a principios de la década de 1970. La noción de Unix de un "proceso"se convirtió en un hilo de control secuencial más un espacio de direcciones virtuales (por cierto, la noción de Unix de un proceso está derivada directamente del diseño del proceso Multics [Saltzer, 66]). Entonces, los "procesos", en el sentido de Unix, son máquinas bastante pesadas. Como no pueden compartir memoria (cada uno tiene su propio espacio de direcciones), interactuando a través de buses de datos, señales, etc. La memoria compartida (también un mecanismo bastante pesado) se agregó mucho más tarde.
Después de un tiempo, los usuarios de Unix comenzaron a perder los viejos procesos que podían compartir memoria. Esto condujo a la ïnvención"de subprocesos: procesos de estilo antiguo que compartían el espacio de direcciones de un único proceso de Unix. También fueron llamados "livianos", en contraste con los procesos "pesados"de Unix. Esta distinción se remonta a finales de los años setenta o principios de los ochenta, es decir, a los primeros "microkernels"(Thoth (precursor del V-kernel y QNX), Amoeba, Chorus, la familia RIG-Accent-Mach, etc.).

2. Tipos de Hilos:

Los hilos se pueden clasificar según su implementación:

2.1 En el espacio de usuario (ULT):

2.1.1 Características:

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 núcleo continua planificando el proceso como una unidad y asignándole un único estado (Listo, bloqueado, etc.).

2.1.2 Ventajas:

• 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.
• Permiten que cada proceso tenga su propio algoritmo de planificación personalizado.
• Un paquete de hilos de usuario se puede implementar en un sistema operativo que no acepte hilos.

2.1.3 Desventajas:

• 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 lasa llamadas al sistema es usando la técnica de jacketing, que es convertir una llamada bloqueante en no bloqueante; esto se consigue comprobando previamente si la llamada al sistema bloqueará el proceso o no. Si es así, se bloquea el hilo y se pasa el control a otro hilo. Más adelante se reactiva el hilo bloqueado y se vuelve a realizar la comprobación, hasta que se pueda realizar la llamada al sistema sin que el proceso completo sea bloqueado.

2.2 A nivel de kernel (KLT):

2.2.1 Características:

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. En otras palabras, el SO es quien crea, planifica y gestiona los hilos. Se reconocen tantos hilos como se hayan creado.

2.2.2 Ventajas:

• 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

2.1.3 Desventajas:

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

¿Cómo se relacionan los hilos a nivel de kernel y los de usuario?

Existen 3 formas para establecer la relación:
• Modelo Mx1 (Many to one):
El modelo asigna múltiples hilos de usuario a un hilo del kernel. Este caso se corresponde a los hilos implementados a nivel de usuario, ya que el sistema solo reconoce un hilo de control para el proceso. Tiene como inconveniente que, si un hilo se bloquea, todo el proceso se bloquea. También, dado que solo un hilo puede acceder al kernel cada vez, no podrán ejecutarse varios hilos en paralelo en múltiples CPUs.
• Modelo 1x1 (one to one):
El modelo asigna cada hilo de usuario a un hilo del kernel. Proporciona una mayor concurrencia que el modelo anterior, permitiendo que se ejecute otro hilo si uno se bloqueó. Tiene como inconveniente que cada vez que se crea un hilo a nivel de usuario, se crea un hilo a nivel del kernel, y la cantidad de hilos a nivel del kernel están restringidos

en la mayoría de los sistemas.
• Modelo MxN (many to many):
El modelo multiplexa muchos hilos de usuario sobre un número menor o igual de hilos del kernel. Cada proceso tiene asignado un conjunto de hilos de kernel, independientemente de la cantidad de hilos de usuario que haya creado. No posee ninguno de los inconvenientes de los dos modelos anteriores, ya que saca lo mejor de cada uno. El usuario puede crear tantos hilos como necesite y los hilos de kernel pueden ejecutar en paralelo. Asimismo, cuando un hilo se bloquea, el kernel puede planificar otro hilo para su ejecución. Entonces, el planificador a nivel de usuario asigna los hilos de usuario a los hilos de kernel, y el planificador a nivel de kernel asigna los hilos de kernel a los procesadores.

2.3 Hibridos:

Se han investigado varias formas de tratar de combinar las ventajas de los hilos a nivel de usuario con los hilos a nivel de núcleo. Una forma es usar hilos a nivel de núcleo y luego multiplexar los hilos a nivel de usuario en algunos o todos los hilos del núcleo. En este diseño, el núcleo sólo conoce los hilos del nivel del núcleo y los programa. Algunos de esos hilos pueden tener múltiples hilos a nivel de usuario multiplexados encima de ellos. Estos hilos a nivel de usuario se crean, destruyen y programan igual que los hilos a nivel de usuario en un proceso que se ejecuta en un sistema operativo sin capacidad de multihilo. Este modelo proporciona lo último en flexibilidad.

2.3.1 Ventajas:

• Algunos de estos hilos pueden tener varios hilos de nivel usuario multiplexados encima de ellos; los hilos de nivel de usuario de crean, destruyen y planifican de igual forma que los hilos de nivel usuario en un proceso que se ejecuta en un sistema operativo sin capacidad de multihilamiento.

2.3.2 Desventajas:

• En este modelo cada hilo a nivel de núcleo tiene un conjunto de hilos a nivel de usuario que se turnan para utilizarlo.

Importancia del lenguaje de programación:

El lenguaje de programación importa a la hora de implementar hilos, sobre todo cuando se trata de hilos a nivel de usuario, ya que estos son programados por el usuario en el propio algoritmo que se quiere ejecutar, por lo que el lenguaje utilizado tiene ciertas implicaciones; como la sintaxis específica de cada lenguaje y la utilización de determinadas librerías para la manipulación de estos hilos. También, el lenguaje debe soportar este tipo de programación, llamada multi-threading, pues no todos los lenguajes están pensados o tienen opciones para este tipo de trabajos. También cabe recalcar la importancia del procesador utilizado, siendo un requerimiento a nivel de hardware. Generalmente deberemos contar con un procesador con soporte multi-hilo, y procesamiento multiple; sean, por ejemplo, los procesadores multi-nucleo. Sin embargo, estos no representan una gran dificultad puesto que la mayoría de procesadores en el mercado actual cuentan con esta capacidad. Por otro lado, ciertos procesadores pueden cambiar más rápido que otros entre procesos, o lo que se llama en este ámbito como "contexto", por las características de su reloj interno (oscilador), por lo que con procesadores diferentes podremos obtener resultados o tiempos de respuesta diferentes (sea esta una diferencia ligera o no).

Importancia del hardware:

Tenemos que en un procesador multinucleo se pueden llevar a cabo varias tareas en paralelo, sin embargo, en un núcleo independiente esto no será posible, y ahí es donde entra la concurrencia de las sub-tareas requeridas. En este caso el núcleo alternará entre una tarea y otra; nuestro procesador actuará como un interruptor en constante cambio de estado. Para que tengan lugar estos cambios de estado es preciso aclarar que el procesador debe tener programada una secuencia de cambio; ya sea por unidad de tiempo, por frecuencia o no determinística. Y por supuesto, estos cambios los ejecuta el componente físico (núcleo). Es importante recordar que los hilos no son un componente físico, sino una funcionalidad que fuerza a los núcleos a alternar entre tareas de poca complejidad. Para hacer esto, se basa en los ciclos del CPU y se ve directamente afectado por la arquitectura del procesador, que puede proveer o no de facilidades para este cambio de estado.

Referencias

[1] Rodriguez, E. *Nucleos e Hilos en un microprocesador: Qué son y en qué se diferencian?*, (Julio. 07. 2017), EL ESPAÑOL
https://www.elespanol.com/omicrono/tecnologia/20170707/nucleos-hilos-procesador-diferenci
229478224_0.html

[2] Urzua, M. *Implimentación de Hilos*, (Marzo. 08. 2017), Instituto tecnológico de Tepic.
https://es.slideshare.net/ThemickyeBMO/implementacin-de-hilos

[3] Jackson, C. *¿Qué son los hilos de un procesador?*, www.techlandia.com.
https://techlandia.com/son-hilos-procesador-info_338189/

[4] Miquel, B. *Historia de la informatica* (Mar., 1936).
https://books.google.com.co/books?id=8wwUowhAp_MC&pg=PA19&dq=historia+de+la+
informatica&hl=es&sa=X&ved=2ahUKEwiJmqm_tKrqAhXFhOAKHZnxCXIQ6AEwAXoECAIQAg#v=
onepage&q=historia%20de%20la%20informatica&f=false

[5] Martínez Cano, H. *Arquitectura de máquinas y computadoras curso II,*Universidad Nacional de Ingeniería, Nicaragua, Managua, (Jul., 2007).

[6] Anonimo. *¿Qué son los núcleos e hilos de un procesador?*, (Nov.7.2017), www.grupoinformatico.com.
http://https://www.elgrupoinformatico.com/que-son-los-nucleos-hilos-procesador-t39601.
html

[7] Cheriton, DR. *Estructuración multiproceso y el sistema operativo Thoth , Ph.D. Tesis.* Universidad de Waterloo, (1979).

[8] Lopez, I. *Lenguaje de programación: PL/1*,www.academia.edu, https://www.academia.edu/
32200339/Lenguaje_de_programaci\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\
@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\
egroup\spacefactor\accent@spacefactor\futurelet\@let@token\protect\penalty\@M\
hskip\z@skipn_PL_1

[9] Tanenbaum, S. *Operating System Threads*, (Jan 25, 2002), www.informit.com
https://www.informit.com/articles/article.aspx?p=25075&seqNum=4 (Dec. 5, 2013).

[10] E. Ceruzzi, P. *Breve historia de la computación*
https://books.google.com.co/books?id=eBSGDwAAQBAJ&printsec=frontcover&dq=
historia+de+la+informatica&hl=es&sa=X&ved=2ahUKEwjDx-j40qrqAhWNneAKHQfwBw8Q6AEwAnoECAUQAg
v=onepage&q&f=false.