# Hilos al Nivel del Microprocesador

Tobías García Mejía

17 de Julio del 2020

## ¿Qué es un Hilo?

Antes de hablar de hilos, lo ideal es describir a sus predecesores, los procesos. Los SOs (Sistemas Operativos) están diseñados para coordinar la forma en que múltiples tareas pueden acceder al procesador, es por esto que para ejecutar cada tarea el SO utiliza lo que se conoce como un proceso, que no es más que una secuencia de ejecución de instrucciones que cuenta con dos elementos principales: el código a ejecutar y un conjunto de datos referentes a este, además de otros elementos para caracterizarlo, como un estado, una prioridad, múltiples registros, información sobre el tiempo de procesamiento, etc. donde esta información es almacenada en lo que se conoce como un bloque de control de proceso [1].

Toda esta información extra hace que un proceso sea algo costoso, además de que hay múltiples tareas que quizá no necesiten de todas estas herramientas o que se dese que se ejecuten de forma diferente [2], por lo cual un proceso se subdivide en varios **procesos ligeros** (*lightweight process*) o también conocidos como **hilos**, los cuales permiten ejecutar pequeños fragmentos de código de forma concurrente o incluso paralela[1], aprovechando mucho más los recursos de máquina y propiciando un código más ordenado y estructurado. Una característica importante es que los hilos pertenecientes a un mismo proceso comparte un mismo segmento de memoria, así como los ficheros abiertos, por lo cual la comunicación entre ellos es mucho más eficiente que entre procesos [4].

## Historia de los Hilos

Alrededor del año 1965, el Berkeley Timesharing System fue uno que los primeros sistemas operativos en emplear un sistema de tiempo compartido (*Time Sharing System*), el cual permite que varios usuarios puedan utilizar una misma unidad de procesamiento para realizar sus diferentes tareas [5], por lo cual la noción de hilo, que en ese entonces se denominaba simplemente proceso, se puede remontar hasta aquella época [6].

El lenguaje PL/1 diseñado por IBM sobre el cual se desarrollaba Multics, otro de los primeros sistema operativos que también empleaba la modalidad de tiempo compartido [7], contaba con un constructor que permitía la creación de hilos (a los cuales se le refería como *Tasks*), aunque se decidió que estos no podían asignarse a un proceso porque no había protección entre los hilos de control, por lo cual la idea de utilizar estos *Tasks* fue suprimida del lenguaje PL/1 por parte de IBM [6].

Más adelante, alrededor de los 1970, la llegada de Unix trajo consigo su propia noción de proceso, la cual consistía, además de un hilo de control secuencial, un espacio virtual de direcciones, haciendo que un proceso fuera algo de gran demanda de recursos, además de como no compartían un mismo espacio de memoria entre ellos, su comunicación debía ser mediante señales o tuberías (*pipelines*)[2] [6].

No pasó mucho tiempo hasta que los usuarios de Unix necesitaron de nuevo la ventaja de poder compartir los espacios de memoria dentro de los procesos, de donde la idea de hilo tomó lugar, procesos livianos que compartieran un mismo espacio de memoria dentro de un mismo proceso Unix [6].

## Tipos de Hilos

Existen dos tipos de hilos, los de nivel de usuario (ULT *user level thread*) y los de nivel de kernel (KLT *kernel level thread*).

Los ULT son aquellos que se emplean al utilizar una librería de hilos para desarrollar una aplicación multihilo, en este caso la librearía se encarga de todo el manejo de estos, desde su creación y destrucción hasta la forma en que se alternará entre los hilos y su comunicación.

[1] En la concurrencia se van ejecutando instrucciones de varias tareas por ciertos periodos, mientras que en el paralelismo se aprovechan varios núcleos del procesador para ejecutar varias tareas literalmente al mismo tiempo [3]

[2] Es una forma de procesar datos tal que la salida de un proceso es justo la entrada del siguiente [8]

Como la librearía es la que realiza la gestión de los hilos, su implementación es independiente del SO, lo cual es una gran ventaja respecto a la portabilidad, pero a su vez una gran desventaja respecto al uso de los recursos de máquina, pues el SO no puede reconocer el uso de múltiples hilos, lo cual deriva en que el proceso sea tratado como si tuviera un único hilo, es decir, todos sus hilos como un todo son clasificados dentro de uno de los estados de un proceso[3], lo cual implica que si uno de los hilos realiza una llamada bloqueante al sistema[4], todo el proceso será bloqueado, y se le indicará al procesador que continúe con otra tarea diferente en lugar de sólo bloquear el hilo que realizó la llamada, y procesar otro relativo al mismo proceso [1].

Una ventaja importante de los ULTs es que se tiene un grado de control mucho mayor respecto a la forma en que los hilos van a implementarse, como se menciona en [2], en particular se puede determinar cómo será el planificador[5] que gestione la ejecución de los hilos, algo que viene predeterminado por el SO dentro de los KLTs. Otra ventaja es que el cambio de hilo no requiere privilegios de modo núcleo, lo cual ahorra el coste de tener que cambiar del modo usuario al modo núcleo [1].

Por otra parte, dos desventajas importantes de los ULTs son que, como se mencionó antes, gracias a que el SO no detecta el proceso como multihilo, el programa no puede hacer uso de múltiples núcleos de procesamiento en caso de que se cuente con varios, eliminando la posibilidad del paralelismo, mientras que la otra desventaja es la que se había mencionado antes respecto al posible bloqueo de todo el proceso [1].

En contraste, los KLTs son administrados netamente por el kernel, en donde este reconoce el programa como un solo proceso pero de múltiples hilos, lo cual permite que el planificador por defecto sea capaz de asignar diferentes hilos de este mismo proceso a diferentes núcleos de procesamiento, aprovechando mejor los recursos de máquina, además de que si uno de los hilos entra en el estado bloqueado, el kernel puede continuar con otro hilo relativo mismo proceso [1].

Sin embargo, utilizar KLTs también acarrea desventajas, como por ejemplo para pasar de un hilo a otro se requiere entrar en el modo núcleo para tener los privilegios, lo cual lleva un coste adicional. Utilizar ULTs prohíbe el uso de varios núcleos de procesamiento al tiempo, pero resulta menos costoso moverse entre hilos, mientas que al utilizar KLTs sucede todo lo contrario; la elección de cual tipo de hilo utilizar depende intrínsecamente de la aplicación que se vaya a desarrollar [1].

Para finalizar esta sección, no está demás mencionar que existen algunos SOs que cuentan con combinaciones entre los ULTs y los KLTs, como Solaris, en donde los hilos se crean como ULTs, pero estos con asociados a un número menor o igual de KLTs [1].

## Implementación de Hilos a Nivel de Hardware

La implementación de los hilos no requiere de una herramienta de hardware adicional al procesador y la memoria, pues es una gestión netamente hecha por el SO, por lo cual en esta sección se discutirá cómo este administra los hilos.

Un hilo, al igual que un proceso, en cualquier momento se encuentra en uno de tres estados [9]:

- **Ejecutandose:** Cuando se está haciendo uso del procesador para avanzar en la ejecución.
- **Bloqueado:** No se puede continuar con la ejecución porque depende de un resultado o una I/O.
- **Listo:** Está recién creado o listo para utilizar el procesador de nuevo luego de que fuera bloqueado.

Entre estos tres estados existen cuatro posibles transiciones [1]:

- **Creación:** Cuando se crea un proceso, este se crea con un único hilo, el cual puede crear al resto. Cuando se crea un hilo, el hilo que lo creó le entrega un puntero a las instrucciones que debe seguir, le otorga su propio contexto y espacio de pila.

[3] Un proceso puede encontrarse en uno de cinco estados, ejecutando, listo, bloqueado, nuevo y saliente. En particular, que un proceso se encuentre en estado bloqueado significa que no puede continuar porque está esperando algo, como una operación I/O o a que otro hilo o proceso termine su ejecución [1].

[4] Consiste en una llamada al sistema operativo que deriva en que este coloque al hilo o proceso en el estado bloqueado.

[5] Es el encargado de distribuir el tiempo del procesador para ejecutar los diferentes procesos o hilos.

- **Bloqueo:** Cuando el hilo necesita un dato o esperar hasta que suceda un evento, este se bloquea, y se almacena en memoria los registros de usuario, el contador de programa y punteros de pila.
- **Desbloqueo:** Cuando un hilo se encuentra bloqueado y el dato que necesita ya está preparado o el evento por el que esperaba sucedió, el hilo entra de nuevo en el estado listo.
- **Finalización:** Cuando el hilo termina de ejecutar las instrucciones especificadas, se libera su registro de contexto y pilas.

Como todos los hilos de un mismo proceso comparten el mismo segmento de memoria y ficheros abiertos, es de vital importancia la forma en que se sincroniza su ejecución para poder utilizar la concurrencia o el paralelismo, sin que se presenten problemas respecto a la lectura y escritura de información en la memoria. Para la concurrencia en particular existen tres métodos principales: el de semáforos, el de monitores y el de transferencia de mensajes, cuyos detalles e implementaciones son discutidos en [1] y [9].

Para finalizar esta sección, hay una última observación que es importante hacer, y es que la implementación de aplicaciones multihilo no depende en sí del hardware utilizado, sin embargo la forma en que se ejecutan sí depende de la cantidad de núcleos con los que cuente el procesador, respecto al poder utilizar la técnica de paralelismo en lugar de simple concurrencia, por lo cual a manera de conclusión, la implementación de una aplicación multihilo no dependen del hardware empleado, aunque la forma en que el SO ejecuta los diferentes hilos sí, pues esta última depende del número de núcleos del procesador.

# Implementación de Hilos a Nivel de Software

Como se mencionó antes, para el desarrollo de una aplicación multihilo se requiere del apoyo de una librería especializada para esta tarea, en este caso se tomará como ejemplo el lenguaje C, en donde la librería a incluir es *pthread.h*[6].

Cada hilo debe contar con su propio identificador, para ello se debe declarar una variable de clase **pthread_t**. Para crea un hilo se utiliza la función **pthread_create**(), la cual posee cuatro parámetros, el primero es el identificar del hilo en cuestión, debe ser tipo **pthread_t***; el segundo corresponde a los atributos del hilo como puede ser por ejemplo su nivel de prioridad, si se deja en *NULL* el hilo se creará con los atributos por defecto[7], debe ser de tipo **pthread_attr_t***; el tercero corresponde a la referencia de la función que contiene las instrucciones que se ejecutarán dentro del hilo, debe ser de tipo **void***(*)(**void***), el cual no es más que una función que recibe un puntero **void*** y que devuelve un puntero **void***, por lo tanto cuando se implemente esta función se debe cumplir con este detalle; y el último es el argumento que se le pasará a la función del tercer parámetro al comenzar a ejecutarla, de esta forma, es posible pasar a la función lo que se necesite y dentro de esta realizar un *cast* para recuperar la clase del objeto [10]. Esta función retorna 0 si el hilo se creó con éxito, o un valor distinto en caso contrario.

Si se crea un hilo dentro de **main**() y se corre el código se verá que no se ejecutan las instrucciones del hilo, esto es pues como el hilo principal del proceso continua con su ejecución, al no tener más instrucciones termina la ejecución de todo el programa, haciendo que el hilo que se creó no tenga tiempo de ejecutarse, por lo cual, de forma más general, para hacer que un hilo espere a la terminación de otro se emplea la función **pthreah_join**(**pthread_t** thread, **void**** rval_ptr) la cual se encarga de que el hilo que invoca esta función espere hasta que el hilo *thread* termine su ejecución, si *rval_ptr* no es *NULL*, en esa dirección se almacena la forma en que termina el hilo *thread*.

Esta sólo fue una breve introducción al manejo de hilos en el lenguaje C, en otros lenguajes de programación la forma de implementar los hilos varía, por lo cual se puede afirmar que el lenguaje de programación es importante a la hora de desarrollar una aplicación multihilo. Para concluir, en esta sección solo se habló de los hilos de la librería Pthreads, aunque existen otras alternativas como Win32 o Java, por lo cual si se desea profundizar más en el tema, se sugiere consultar [12] para más información sobre las librerías de hilos.

[6] Además de incluir la librería, a la hora de compilar el programa desde una terminal se debe agregar *–pthread* para linkear el programa con esta [10]

[7] Si se desea modificar los atributos por defecto se puede crear una variable del tipo **pthread_attr_t**, inicializarla con sus valores por defecto mediante la función **pthread_attr_init**(**pthread_attr_t*** atributes) y luego utilizar otras funciones para modificar cada atributo en particular [11]

# Referencias


[1] W. Stallings, *Sistemas Operativos: Aspectos internos y principios de diseño*, 5ª ed., Alhambra, ed. 2005, ISBN: 8420544620.

[2] B. D. Marsh, M. L. Scott, T. J. LeBlanc y E. P. Markatos, "First-Class User-Level Threads," *ACM SIGOPS Operating Systems Review*, nº 5, sep. de 1991, *First-class user-level threads*.

[3] H. Y. Badgujar. (2013). Concurrency Vs Parallelism, dirección: `https://www.researchgate.net/post/Concurrency_Vs_Parallelism` (visitado 2020).

[4] Tutorials Point. (s.f.). Operating System - Multi-Threading, dirección: `https://www.tutorialspoint.com/operating_system/os_multi_threading.htm` (visitado 2020).

[5] Encyclopedia Britannica. (s.f.). Time-sharing, dirección: `https://www.britannica.com/technology/time-sharing` (visitado 2020).

[6] B. O'Sullivan. (2002). The history of threads, dirección: `http://www.serpentine.com/blog/threads-faq/the-history-of-threads/` (visitado 2020).

[7] M. Rouse. (2005). Multics (Multiplexed Information and Computing Service), dirección: `https://whatis.techtarget.com/definition/Multics-Multiplexed-Information-and-Computing-Service` (visitado 2020).

[8] T. Nuñez. (2018). Arquitectura Pipeline, dirección: `https://www.electrontools.com/Home/WP/modelo-de-arquitectura-pipeline/` (visitado 2020).

[9] A. S. Tanenbaum y A. S. Woodhull, *Sistemas Operativos: Diseño e Implementación*, 2ª ed., P. Hall, ed. 1998, ISBN: 9701701658.

[10] (2007). Procesos e Hilos en C de Unix/Linux, dirección: `http://www.chuidiang.org/clinux/procesos/procesoshilos.php` (visitado 2020).

[11] (2007). Algunos detalles sobre los Threads, dirección: `http://www.chuidiang.org/clinux/procesos/mashilos.php` (visitado 2020).

[12] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 8ª ed., Wiley, ed. 2008, ISBN: 0470128720.