**Hilos**

**¿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 [SOLibroCap3].

Toda esta información extra hace que un proceso sea algo costoso, además de que hay múltiples tareas que quizá no necesite de toda estas herramientas o que se dese que funcionen de forma diferente [UTLProyecto], 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\*, aprovechando mucho más los recursos de máquina y propiciando un código más ordenado y estructurado. Un característica importante es que los hilos pertenecientes 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 [TutorialsPoint].

\*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 [WebParalelismo].

Concurrencia: A grandes rasgos, varias tareas se pueden ejecutar de forma concurrente cuando se va ejecutando instrucciones de cada una por ciertos periodos, haciendo que todas avancen [WebParalelismo].

Paralelismo: Varias tareas se pueden ejecutar en paralelo cuando literalmente se ejecutan simultáneamente en diferentes núcleos dentro del procesador.

Antes de definir lo que es un hilo, primero se abordará un concepto más general, el concepto de proceso. Un procesador es capaz de llevar a cabo múltiples tareas a la vez, simultáneamente como en un procesador multinucleo, o empleando otros métodos como la concurrencia\*, que permitían realizar varias tareas aparentando simultaneidad cuando el procesador sólo contaba con un único núcleo [WebParalelismo]; los sistemas operativos están diseñados para coordinar todas estas tareas, asignándoles recursos de hardware y administrando la forma en que se ejecutan, ahí es donde resulta útil la noción de proceso, el cual consta de dos elementos primordiales: un 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 [SOLibroCap3].

Estos detalles hacen que un proceso se pueda considerar algo macro y ostentoso, pues son utilizados para realizar actividades que requieren muchos recursos como ejecutar una aplicación por ejemplo, sin embargo algunos de estos recursos quizá no sean tan relevantes a la hora de realizar actividades más sencillas [UTLProyecto], por lo cual un proceso es subdivido en lo que se conoce como **hilos**, o también denominados **procesos ligeros** (*lightweight process*), los cuales se encargan de la ejecución de porciones del código, permitiendo dividirlo en tareas más pequeñas que pueden ejecutarse de manera concurrente o incluso paralela\*, aprovechando mejor los recursos de máquina y propiciando un código más ordenado y estructurado, además, los hilos comparten un mismo segmento de memoria así como los ficheros abiertos, por lo cual la comunicación entre ellos es mucho más sencilla que entre sus hermanos mayores, los procesos [TutorialsPoint].

Concurrencia: A grandes rasgos, varias tareas se pueden ejecutar de forma concurrente cuando se va ejecutando instrucciones de cada una por ciertos periodos, haciendo que todas avancen.

Paralelismo: Varias tareas se pueden ejecutar en paralelo cuando literalmente se ejecutan simultáneamente en diferentes núcleos dentro del procesador.

**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.

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 de la aplicación, pero es una gran desventaja respecto al uso de los recursos de máquina, pues el SO no puede detectar el uso de múltiples hilos, lo cual deriva en que trate a la aplicación como un proceso de un único hilo, es decir, lo clasifica como un todo en uno de los estados de un proceso\*, lo cual implica que si uno de los hilos realiza una llamada bloqueante al sistema\*, todo el proceso será bloqueado y se le indicará al procesador que continúe con otro diferente en lugar de sólo bloquear el hilo que realizó la llamada y procesar otro relativo a la misma aplicación que se encuentre en el estado listo [OSLibroCap4].

Estados: 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 [SOLibroCap3].

Llamada bloqueante: Consiste en una llamada al sistema operativo que implica que este coloque al hilo o proceso en el estado bloqueado.

Una ventaja importante es que al utilizar ULTs se tiene un grado de control mucho mayor respecto a la forma en que los hilos van a implementarse como se menciona en [ULTProyecto], en particular se pueden determinar cómo será el planificador que gestione la ejecución de los hilos, algo que viene predeterminado por el SO dentro de los KLTs. Otra ventaja relevante al uso de ULTs es que el cambio de hilo no requiere privilegios de modo núcleo, lo cual ahorra el costo de tener que cambiar del modo usuario al modo núcleo [OSLibroCap4].

Por otra parte, dos desventajas importantes del ULT son que, como se mencionó antes, gracias a que el SO no detecta el proceso como multihilo, el programa no puede hacer uso de varios núcleos de procesamiento en caso de que se cuente con más de uno, eliminando así la posibilidad del paralelismo, mientras que la otra desventaja es la que se había mencionado antes, respecto al bloqueo de todo el proceso al momento en que un hilo realiza una llamada bloqueante al sistema [OSLibroCap4].

En contraste, los KLTs son administrados netamente por el kernel, por lo cual este reconoce el programa como un solo proceso pero de múltiples hilos, lo cual permite que el planificador por defecto del kernel sea capaz de asignar diferentes hilos de este mismo proceso a diferentes núcleos de procesamiento, permitiendo el paralelismo y el mejor aprovechamiento de los recursos de máquina, además de que si uno de los hilos es bloqueado, el kernel puede continuar con otro hilo del mismo proceso [OSLibroCap4].

Sin embargo, utilizar KLTs también acarrea sus propias desventajas, como por ejemplo que para pasar de un hilo a otro se requiere entrar en el modo núcleo para tener los privilegios, lo cual lleva un costo adicional. Así, 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, por lo cual la elección de cual tipo de hilo utilizar depende intrínsecamente de la aplicación que se vaya a desarrollar [OSLibroCap4].

Antes de 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 [OSLibroCap4].

Citar también el blog de Javier.

Los KLT dependen del sistema operativo, mientras que los ULT dependen del lenguaje, supongo. Dependen del hardware, para hacer multinucleo o no.

**¿Cómo se implementan los hilos a nivel de hardware?**

Los procesos al fin y al cabo son una herramienta para que el SO pueda aprovechar el potencial del procesador en todo momento, pasando de realizar una única tarea a aparentar realizar varias simultáneamente, regulando la forma en que estas acceden al procesador, los hilos son simplemente otra herramienta para que el SO mejore la forma en que se realizan estas tareas, sin embargo, su implementación no requieren de herramientas de hardware adicionales al procesador y la memoria, pues es dentro del SO en donde la noción de proceso e hilo tienen sentido, por lo cual en esta sección se discutirá cómo el SO administra los hilos, mientras que se dejará para la sección final cómo se realiza la implementación de hilos desde un lenguaje de programación como C.

Un hilo, al igual que un proceso, en cualquier momento se encuentra en uno de tres estados\*:

* **Ejecutándose**: Cuando se está haciendo uso del procesador para avanzar en la ejecución.
* **Bloqueado**: No se puede continuar con la ejecución porque se está esperando una operación I/O o el resultado de algún otro hilo o proceso.
* **Listo**: El hilo o proceso está listo para utilizar el procesador de nuevo luego de haber sido bloqueado o de que el planificador\* haya decidido que otro hilo o proceso necesita hacer uso del procesador. Cuando se crea un hilo este se coloca en estado listo.

\*Esta información se tomó de [MinixCap2].

\*Planificador: Es el encargado de distribuir el tiempo del procesador para ejecutar los diferentes procesos.

Entre estos tres estados existen cuatro posibles transiciones\*:

* **Creación:** Cuando se crea un proceso, este se crea con un único hilo, el cual puede crear el resto de hilos que necesita el proceso. 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.
* **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. Luego de esto el procesador pasa a ejecutar un hilo diferente, dentro de este mismo proceso o en uno distinto, que se encuentre en estado listo.
* **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.

\*Esta información se tomó de [OSLibroCap4].

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 [OSLibroCap5] y [MinixCap2.2].

Antes de finalizar esta sección, hay un última observación que es importante mencionar, 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 cuenta 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.

**¿Cómo se implementan los 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 tomaremos como ejemplo el lenguaje C, en donde la librería a incluir es pthread.h\*.

Cada hilo debe contar con su propio identificador, para ello se debe declarar una variable que lo identificará utilizando la clase **pthread\_t**. Para crea un hilo utilizamos 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 lo pude ser la prioridad este por ejemplo, si se deja en **NULL** el hilo se creará con los atributos por defecto\*, 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\***), la cual no es más que una función que recibe un puntero **void\*** y que devuelve un puntero **void\***, por lo cual cuando se implemente esta función debemos asegurarnos de 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 el **cast** necesario [LaMar]. Esta función retorna 0 si el hilo se creó con éxito, o un valor distinto en caso contrario.

**pthread\_create**(**pthread\_t\* restrict** thread, **const pthread\_attr\_t\* restrict** attributes,

**void\*** (**\***thread\_instructions) (**void\***), **void\* restrict** argument)\*

Donde *thread* es el identificar del hilo en cuestión, *attributes* corresponde a los atributos del hilo, como lo pude ser la prioridad este por ejemplo, si se deja en NULL el hilo se creará con los atributos por defecto\*; *thread\_instructions* corresponde a la referencia de la función que contiene las instrucciones que se ejecutarán dentro del hilo, notemos que el tipo de dato corresponde a una función que recibe un puntero **void\*** y que devuelve un puntero **void\***, por lo cual cuando se implemente esta función debemos asegurarnos de cumplir con este detalle; y finalmente a*rgument* es el argumento que se le pasará a la función *thread\_instructions* cuando esta se comience a ejecutar, de esta forma es posible pasar a la función lo que se necesite y dentro de esta realizar el **cast** necesario [LaMar]. 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 utilizamos la función **pthreah\_join**(**pthread\_t** thread, **void\*\*** rval\_ptr) que se encarga de que el hilo que invoca la 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 [OSIngles] para más información sobre las librerías de hilos.

\*Pthread: 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 [Linker].

\*Pthread\_create: El prototipo de la función es específica en [POSIX], mientras que una más clara explicación de sus parámetros de entrada se brinda en [LaMar].

\*Defecto: 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 **pthread\_attr\_init**(**pthread\_attr\_t**\* atributes) y luego utilizar otras funciones para modificar cada atributo en particular [Attr].

clase **pthread\_**t. La forma de identificar los hilos depende del SO, por lo cual para comparar dos hilos se utiliza la función **pthread\_equal**(**pthread\_t** thread1, **pthread\_t** thread2), la cual retorna 0 si son diferentes y diferente de 0 si son iguales [POSIX].

**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 [Enciclopedia], por lo cual la noción de hilo, que en ese entonces se denominaba simplemente proceso, se puede remontar hasta aquella época [Sullivan].

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 [Tech], 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 [Sullivan].

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, como no compartían un mismo espacio de memoria entre ellos, su comunicación debía ser mediante señales o tuberías\* (*pipelines*) [Sullivan].

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 [Sullivan].

\*Tuberías: Es una forma de procesar datos tal que la salida de un proceso es justo la entrada del siguiente [ArqPiepline].