Sistemas Operativos

Unidad 3: Procesos e Hilos

"La programación secuencial es realmente difícil, y la programación paralela es un paso más allá de eso".

~ Andrew S. Tanenbaum

Procesos e Hilos

Introducción

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones, recursos asociados y un solo hilo de ejecución. De hecho, esa es casi la definición de un proceso. Sin embargo, hay situaciones en las que es deseable tener múltiples hilos de ejecución en el mismo espacio de direcciones ejecutándose casi en paralelo, como si fueran procesos separados (excepto por el espacio de direcciones compartido).

La pregunta que podría surgir es ¿Para qué necesitamos múltiples hilos de ejecución dentro del mismo proceso?

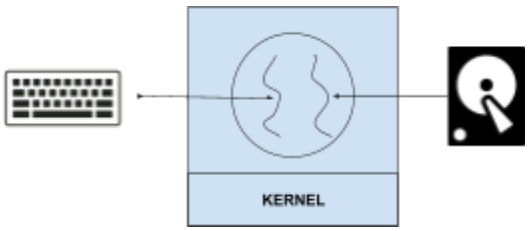
La razón principal es que en muchas aplicaciones necesitamos que varias acciones se ejecuten al mismo tiempo, es decir, que un proceso realice múltiples tareas y que a su vez puedan compartir el espacio de memoria. Lograr esto con múltiples hilos de ejecución dentro del mismo proceso es más simple que tratar de alcanzar la misma solución con procesos separados.

La segunda razón, recae en el análisis de rendimiento. Crear, eliminar o intercambiar un hilo es mucho más rápido que su equivalente operación con procesos, aproximadamente 10 a 100 veces.

La tercera razón tiene que ver también con cuestiones de mejora de rendimiento. Si varios hilos de ejecución realizan muchas operaciones de I/O (entrada/salida), se podrán paralelizar las operaciones aprovechando mejor el uso del CPU.

Algunos ejemplos de aplicaciones que utilizan hilos, pueden ser:

a. Un procesador de texto que realiza 2 tareas básicas. Por un lado, lee los comandos introducidos por el usuario y por otro lanza una tarea de guardado preventivo de los datos. Imaginemos que en caso de no utilizar hilos, podría pasar que el usuario intente utilizar el procesador de texto cuando se está llevando a cabo la tarea de guardado preventivo de datos, por lo que no podría atender la entrada del usuario, que sin lugar a dudas, notaría una baja de rendimiento del programa.



b. Podemos pensar también en un servidor de páginas web, en el cuál se cree un hilo de ejecución para responder a cada pedido de página (request). Imaginemos que este servidor trabaja con una memoria interna similar a una caché donde, cada vez que devuelve una página, la guarda en dicha memoria caché. Un pedido posterior, genera que se cree otro hilo, que puede verificar si la página solicitada se encuentra en el espacio de memoria del proceso (servidor web) y en caso de que se encuentre la devuelve. Por el contrario, si la página no se encuentra en el espacio de memoria del proceso, el hilo en cuestión deberá ir a buscarla al disco. Esto generará que el hilo pase a estado bloqueado y se dará lugar a la ejecución de otros hilos

Otros contextos en los que es conveniente la utilización de hilos son:

- Aplicaciones que trabajen en primer y segundo plano.
- Aplicaciones que proporcionen procesamiento asincrónico.
- Aplicaciones que requieran procesar gran volumen de información y no perder rendimiento.

Cabe destacar que el uso de hilos en aplicaciones es realmente útil, o mejor dicho aprovechable, cuando se dispone de varios núcleos o procesadores.

Definición

Como mencionamos anteriormente, en el enfoque tradicional, tal como lo estudiamos hasta ahora, vimos que un proceso consta de 2 características principales:

1. **Propiedad de recursos**: un proceso incluye un espacio de direcciones de memoria para el manejo de la imagen del proceso (programa, datos, pila y PCB)
2. **Planificación / ejecución**: un proceso puede ser planificado y activado (suspendido, bloqueado, etc) por el sistema operativo.

Tradicionalmente estas 2 características son la esencia de un proceso; pero estas características podrían ser tratadas como independientes una de la otra por el sistema operativo. Esto es lo que hacen muchos (o la mayoría) de los sistemas operativos modernos. Para distinguir entre ambas

características, la unidad que se planifica/activa lo denominaremos **hilo (thread)**, o **proceso liviano**, mientras que la unidad que posee los recursos se suele denominar **proceso.**

Lo que el concepto de hilo añade al modelo ya estudiado de proceso es permitir que se realicen distintas tareas, que podrían ser independientes entre sí, en el mismo espacio del proceso.

Multihilo

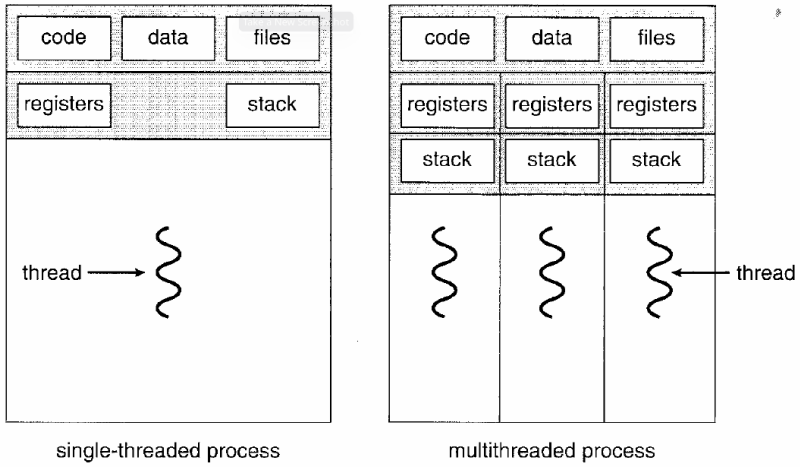
Definición	Es la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso.

Al enfoque tradicional de un solo hilo de ejecución por proceso, se lo conoce como estrategia **monohilo**, en cambio una estrategia multihilo es aquella que soporta múltiples hilos por proceso.

Dentro de un proceso, puede haber uno o más hilos, cada uno con:

- Su código
- Registros (estado de ejecución, pc, etc)
- Una pila de ejecución
- Espacio de almacenamiento para variables locales.
- Acceso a memoria y recursos de su proceso. Esto es compartido con todos los hilos del mismo proceso.

El siguiente gráfico trata de poner de manifiesto las diferencias entre un entorno monohilo y un entorno multihilo:



En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Podemos tener distintos esquemas o configuraciones al respecto:

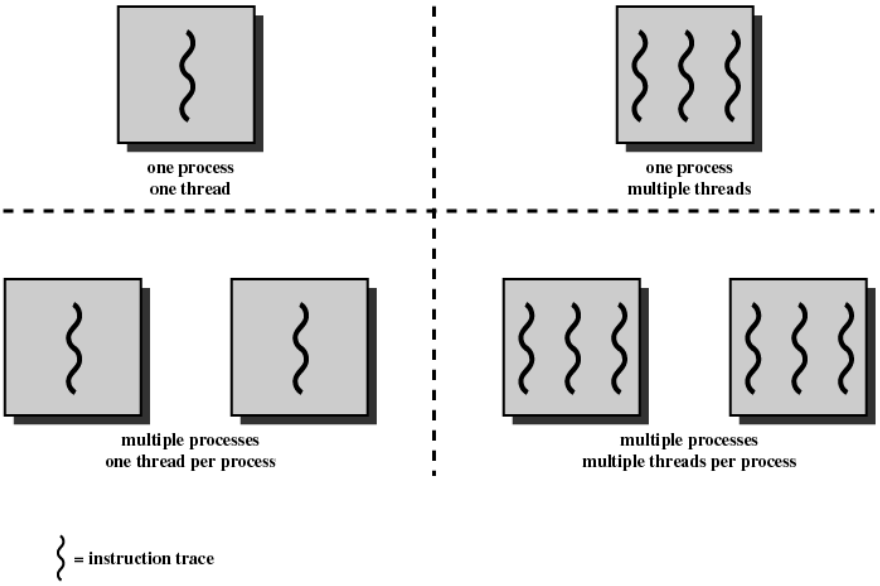


Figure 4.1 Threads and Processes [ANDE97]

Como vemos en la figura anterior, las representaciones que se encuentran a la izquierda de la línea punteada vertical, son procesos con un único hilo, mientras que las representaciones a derecha son procesos con múltiples hilos.

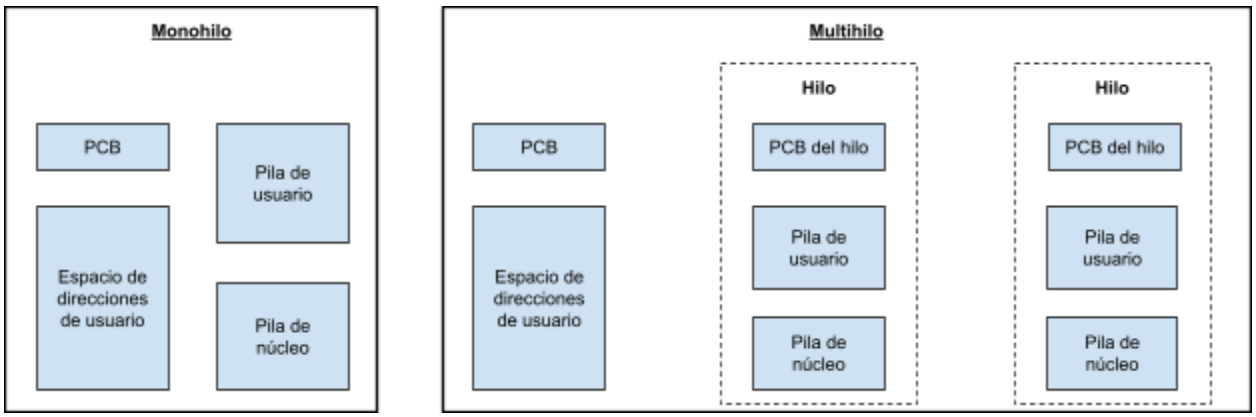
Los sistemas UNIX tradicionales, eran sistemas operativos multiprogramados pero que sólo soportaban un único hilo por proceso (monohilo).

Cuando se ejecuta un proceso con múltiples hilos en un sistema con un único CPU, la ejecución de los hilos entrelaza en el tiempo, similar a lo visto con multiprogramación y procesos. El sistema da la ilusión de procesos/hilos que se ejecutan en paralelo.

Diferencias entre proceso e hilo

En un modelo monohilo, la imagen del proceso incluye el PCB, espacio de direcciones de usuario y la pila de usuario y núcleo para las llamadas y retornos de funciones.

En un entorno multihilo, sigue existiendo un único PCB un un espacio de direcciones de usuario asociado al proceso, pero aparecen múltiples pilas y múltiples PCB uno por cada hilo del proceso.



Del gráfico anterior se observa que todos los hilos de un mismo proceso comparten el estado y los recursos del proceso, residen en el mismo espacio de memoria y tienen acceso a los mismos datos; con lo cual, si un hilo modifica algún dato, todo el resto de los hilos tienen acceso al dato modificado.

De lo expuesto, se deriva que no existe un mecanismo de protección entre hilos como sí lo hay entre procesos independientes. Esto se debe a que:

1. Todos los hilos de un proceso siempre pertenecen al mismo usuario, que es dueño del proceso. Distinto a lo que ocurre con procesos donde cada proceso puede pertenecer a un usuario distinto.
2. Todos los hilos de un proceso comparten el mismo espacio de memoria.

En la siguiente tabla se especifican los ítems compartidos entre los distintos hilos de un mismo proceso y los ítems propios (privados) de cada hilo:

Ítems por proceso	Ítems por hilo
• Espacio de memoria (address space) • Variables globales • Punteros a archivos • Punteros a procesos hijos • Señales • Información estadística	• PC (program counter) • Registros • Stack • Estado

Ventajas de usar hilos

Los mayores beneficios provienen de las mejoras en el rendimiento:

1. Es más rápido crear un nuevo hilo dentro de un proceso existente que un nuevo proceso.
2. Es más rápido finalizar un hilo que un proceso.
3. Es más rápido el cambio entre hilos que entre procesos.
4. Los hilos mejoran la eficiencia de la comunicación entre tareas. La comunicación entre procesos independientes requiere de la intervención del sistema operativo, en un entorno multihilo, al estar los hilos dentro del mismo proceso, la comunicación se puede realizar sin intervención del sistema operativo.

Estados de los hilos

En un sistema operativo multihilo, la planificación se realiza a nivel de hilo. Sin embargo, existen diversas acciones que afectan el estado de todos los hilos de un proceso, por ejemplo suspender o finalizar un proceso implica que todos sus hilos sean suspendidos o finalizados respectivamente.

Los principales estados que se manejan a nivel de hilos son:

- **Creación**: cuando se crea un proceso también se crea un hilo. Luego se pueden crear otros hilos y cada uno tendrá su propio contexto, pila, etc.
- **Bloqueo**: cuando un hilo espera por algún evento, se bloquea, almacenando los registros de usuario. El procesador puede ejecutar otro hilo en estado Listo dentro del mismo proceso o en otro.
- **Desbloqueo**: cuando ocurre el evento por el cual el hilo bloqueado espera, el mismo pasa a la cola de listos.
- **Finalización**: cuando se completa un hilo, se liberan sus registros

No tiene sentido estados como Suspendido, ya que son conceptos que se aplican a nivel de proceso.

Sincronización de hilos

Cuando desarrollamos aplicaciones utilizando hilos, contamos con beneficios en cuanto al rendimiento de la ejecución de nuestro programa, gracias a que parte de nuestra aplicación se ejecutará de forma solapada. Pero la concurrencia también acarrea algunos problemas como son la condición de carrera o el deadlock. Por suerte, existen técnicas que nos ayudan a evitarlos, Veremos estos problemas y posibles soluciones en las próximas unidades.

Hilos en el espacio de usuario o en el espacio del kernel

Existen principalmente dos formas de implementar hilos: una son los llamados hilos en el espacio de usuario y otra son los hilos en el espacio del kernel, aunque también es posible una implementación híbrida.

Cualquiera de las formas mencionadas requieren de una biblioteca que provea las funciones correspondientes para gestionar los hilos.

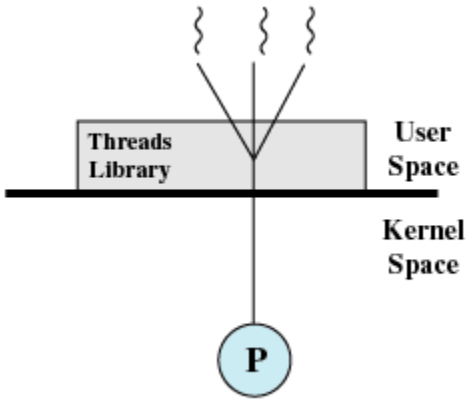
Una biblioteca de hilos proporciona al programador una interfaz para gestionar los hilos. Continuando con lo mencionado en el primer párrafo, dicha biblioteca puede implementarse en el espacio de usuario o puede ser el sistema operativo quien la provea, es decir, implementada a nivel del kernel. La principal diferencia entre estos dos enfoques es que en el primer caso, para gestionar hilos se invocarán funciones dentro del mismo programa, mientras que en el segundo caso serán llamadas al sistema (syscall).

Existen principalmente 3 bibliotecas para manejo de hilos:

1. POSIX Pthreads: puede implementarse tanto a nivel usuario como kernel
2. Win32: sólo disponible en sistemas Windows y a nivel kernel.
3. Java: utiliza la biblioteca del sistema host, por lo que puede utilizar POSIX o Win32.

A continuación describiremos los dos métodos mencionados para implementar hilos, junto con sus ventajas y desventajas.

Hilos en el espacio de usuario o ULT



El enfoque ULT (*User Level Thread*) puro consiste en gestionar los hilos completamente en el espacio del usuario, es decir desde el proceso que los genera. El kernel no gestiona los hilos, de

hecho no conoce su existencia. En lo que respecta al kernel, está gestionando procesos ordinarios de un solo hilo.

La ventaja es que se puede implementar hilos en un sistema operativo que no admite hilos. Esto se logra a través del uso de bibliotecas.

Todos los sistemas operativos tradicionales solían caer en esta categoría.

Por defecto, una aplicación comienza con un solo hilo de ejecución. A partir de ese instante, el proceso puede crear nuevos hilos utilizando llamadas a la biblioteca de hilos. La aplicación cede el control a la biblioteca de hilos y una vez creado el hilo, el control es retornado al hilo principal de ejecución de la aplicación. Es responsabilidad de la aplicación implementar un algoritmo de planificación de hilos para determinar cual de todos los hilos creados, que se encuentren en estado *listo* pasará a ejecutarse.

Como vemos, todo lo descrito sucede dentro del espacio de usuario y dentro de un proceso. El kernel continúa planificando el proceso como una unidad y asigna al proceso un único estado.

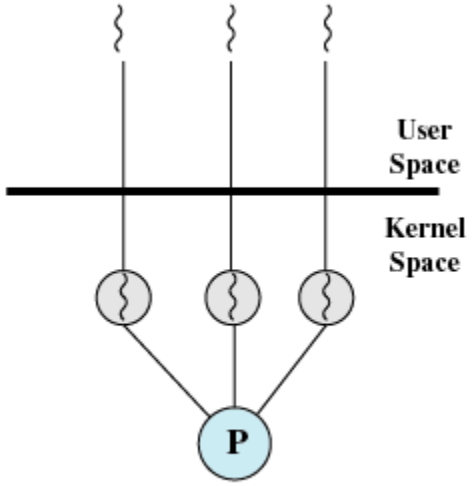
Ventajas de utilizar hilos a nivel de usuario:

1. El cambio entre hilos del mismo proceso no requiere intervención del núcleo, por lo que no se realiza un doble cambio de modo para gestionar los hilos.
2. El algoritmo de planificación puede especificarse dentro de la aplicación.
3. Los ULT son portables, es decir, pueden ejecutarse en cualquier sistema operativo.

Desventaja de utilizar hilos a nivel de usuario:

1. Cuando un hilo realiza una llamada al sistema, si la misma es bloqueante, resulta en el bloqueo del proceso completo.

Hilos en el espacio del kernel o KLT



(b) Pure kernel-level

En un entorno KLT puro, toda la gestión de hilos la realiza el kernel, no hay código de gestión de hilos en el proceso de usuario. Sólo existe una API para acceder a las utilidades del sistema operativo para gestión de hilos.

Al igual que en el enfoque ULT, cualquier aplicación puede programarse para utilizar hilos. Cuando una aplicación necesita crear un hilo, realiza una llamada al sistema, lo que es considerablemente más costoso que en el enfoque ULT debido a que debe intervenir el sistema operativo.

La principal ventaja de este enfoque reside en que si un hilo ejecuta una acción bloqueante, el sistema operativo puede planificar otro hilo del mismo proceso para que comience a ejecutar

Como mencionamos anteriormente, la principal desventaja de utilizar el enfoque KLT en comparación con el enfoque ULT, es que un cambio de hilo del mismo proceso conlleva un cambio de modo.

A modo de resúmen, podemos decir que si bien hay una ganancia significativa entre el uso de ULT respecto de los KLT, si los hilos realizan muchas operaciones que requieren acceso a modo núcleo, esta ganancia no sería tan superior al enfoque KLT.

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue