**Hilos - Threads**

**Definición**

* Características que le permite a una aplicación realizar varias tareas a la vez (**para el usuario, la ejecución es concurrente**). Es una tarea que puede ser ejecutada en paralelo con otra tarea. **INDEPENDIENTES**

**La idea de los hilos, es poder ayudar a los procesos, para que estos puedan finalizar antes de tiempo. Creando dentro de la ejecución de un proceso varios hilos. Asignándoles a cada uno de estos hilos una tarea diferente. Los procesos están más tiempo en E/S, bloqueados, que realmente ejecutando en el sistema.**

**La ejecución es concurrente, seudo paralela. Un procesador, un proceso ejecutando. Un proceso cuando se ejecuta al menos tiene un hilo. Lo que permite agilizar la ejecución es que si hay 3 hilos de un proceso y un hilo pasa a bloqueado, el proceso puede decidir, ejecutar otro hilo, esto es lo que permite, que finalice antes.**

* La mayoría de los SO modernos proporcionan procesos con múltiples secuencias o hilos de control en su interior.

**Agilizar la ejecución, y minimizar el uso de la memoria.**

**Programa, proceso e hilo. Programa entidad pasiva, cuando se ejecuta, entra en actividad se convierte en proceso, un programa puede tener min. 1 proceso. los procesos siempre que estén en ejecución min. Van a tener un hilo. Pueden tener mas de un hilo es una división de tarea, que lo ayuden a finalizar.**

* Se lo considera una unidad básica de utilización de la CPU.

**Unidad básica, siempre q se este ejecutando tiene que estar en CPU. Min se va a tomar 1 hilo. Cuando se ejecuta un proceso como min hay un hilo.**

* Cada hilo posee identificador del thread contador de programa, conjunto de registros y pila.

**Si un proceso tiene un ID, y este proceso utiliza 2 hilos, cada uno de estos va a tener su propio identificador. Un contador de programa de cada hilo, nos va indicar que instrucción se está ejecutando y cual es la que sigue. Conj. Registros del hilo. Pila, se guardan todas las variables del hilo.**

* Comparten con el resto de los hilos del proceso mapa de memoria, ficheros abiertos y señales, semáforos y temporizadores.

**El mapa de memoria del proceso es el mismo, para los hilos que contenga. Temporizador tiempos de ejecución max. Que tiene un proceso va a ser dividido entre los hilos. Compartido para todos los hilos. A diferencia de la prioridad, un proceso puede tener una prioridad, pero los hilos dentro de ese proceso pueden tener diferentes prioridades.**

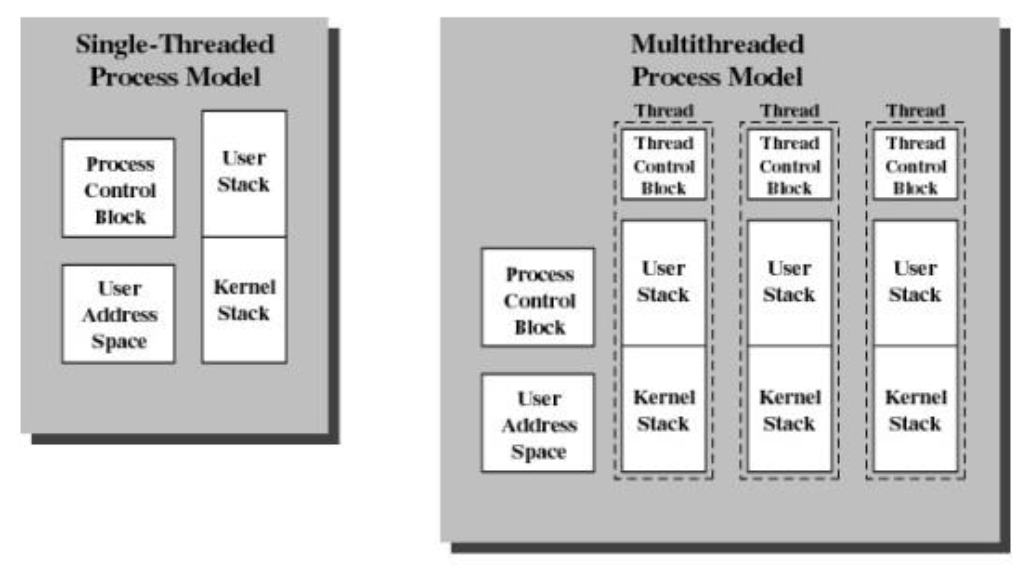
Los mayores **beneficios** de los hilos provienen de las **consecuencias del rendimiento**:

1. Lleva mucho menos tiempo crear un nuevo hilo en un proceso existente que crear un proceso totalmente nuevo. **Cuando creo un proceso, reservo memoria para ese proceso. entonces los hilos que voy creando van a ejecutar instrucciones del proceso que ya está en memoria. Creo proceso cargar memoria, BCP, verificar que tenga la memoria suficiente para ejecutarlo. Cuando se crea un hilo, el proceso ya está en MP. Por eso es más fácil darle al hilo la tarea que tenga que ejecutar.**

2. Lleva menos tiempo finalizar un hilo que un proceso.

3. Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso. **Cambio de modo usuario modo núcleo, cambio de contexto. Son mas rápidos entre hilos que entre procesos.**

4. Los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando. **Que hilos se debe ejecutar primero cual no, los datos que se pasan entre hilos, la sincronización.**



**A la izquierda tenemos un proceso con un hilo. BCP, espacio de direcciones del usuario, las pilas (de ejecución de modo usuario, y modo Kernel)**

**A la derecha tenemos un modelo de proceso multi-hilos. BCP, Dirección de espacio de usuarios. Luego para cada uno de los hilos hay un BCT (estado, id, prioridad, contador de programa), pilas en modo usuario y modo núcleo.**

**Ejemplo. Con una receta. Programa asado, la acción es el proceso.**

**Asado (costillar a la cruz, chinchulines a la parrilla, ensalada (mixta, papa y huevo))El asado es el proceso, los hilos de ejecución son las tareas tareas independientes, para poder lograr el proceso.**

**Hilos-Threads Ejemplos**

• **Servidor de archivos.** Cada vez que llega una nueva petición de archivo, el programa de gestión de archivos puede ejecutar un nuevo hilo. Ya que un servidor manejará muchas peticiones, se crearán y finalizarán muchos hilos en un corto periodo de tiempo. Además, ya que los procesos o los hilos en un servidor de archivos deben compartir archivos de datos y coordinar sus acciones, es más rápido usar hilos y memoria compartida que usar procesos y paso de mensajes para esta coordinación.

• **Trabajo en primer plano y en segundo plano.** Por ejemplo, en un programa de hoja de cálculo, un hilo podría mostrar menús y leer la entrada de usuario, mientras otro hilo ejecuta los mandatos de usuario y actualiza la hoja de cálculo. Esta forma de trabajo a menudo incrementa la velocidad que se percibe de la aplicación, permitiendo al programa solicitar el siguiente mandato antes de que el mandato anterior esté completado**. El auto guardado, son hilos.**

• **Procesamiento asíncrono.** Los elementos asíncronos de un programa se pueden implementar como hilos. Por ejemplo, se puede diseñar un procesador de textos con protección contra un fallo de corriente que escriba el buffer de su memoria RAM a disco una vez por minuto. Se puede crear un hilo cuyo único trabajo sea crear una copia de seguridad periódicamente y que se planifique directamente a través del sistema operativo; no se necesita código adicional en el programa principal que proporcione control de tiempo o que coordine la entrada/salida. **Hilos que se ejecutan en una cantidad de tiempo de forma asíncronos, corte de energía, luego trae la ultima versión recuperación.**

• **Velocidad de ejecución.** Un proceso multihilo puede computar una serie de datos mientras que lee los siguientes de un dispositivo. En un sistema multiprocesador pueden estar ejecutando simultáneamente múltiples hilos de un mismo proceso. De esta forma, aunque un hilo pueda estar bloqueado por una operación de E/S mientras lee datos, otro hilo puede estar ejecutando.

• **Estructura modular de programas.** Los programas que realizan diversas tareas (**independientes y si son dependientes tiene que tener una sincronización que lo vemos en concurrencia**) o que tienen varias fuentes y destinos de entrada y salida, se pueden diseñar e implementar más fácilmente usando hilos.

**Estados de un Hilo-Thread**

Un Thread puede estar en cualquiera de estos 3 estados:

**Ejecución**: tiene en ese momento la CPU y está activo.

**Bloqueado**: espera que algún proceso o hilo lo desbloquee.

**Listo**: está planificado para ejecutarse y lo hace apenas llegue su turno.



**Si un proceso está suspendido, los hilos dejan de correr en MP, todos los hilos son descargadas en MS. No hay estado de suspendido de hilos.**

**Hilos Nivel Usuario ULT–UserLevelThreads**

* La aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos.

**Se pueden crear muchos hilos a nivel usuario y los hilos que se están ejecutando a nivel núcleo no son conscientes, ni tiene relación con esos hilos que se están creando / ejecutando a nivel usuario**.

* Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos.

**O librería, para poder crear, ejecutar, coordinar y gestionar todo lo relacionado con los hilos.**

* La biblioteca de hilos contiene código para la creación y destrucción de hilos, para paso de mensajes y datos entre los hilos, para planificar la ejecución de los hilos, y para guardar y restaurar el contexto de los hilos.

**Ventajas:**

* El cambio de hilo no requiere privilegios de modo núcleo porque todas las estructuras de datos de gestión de hilos están en el espacio de direcciones de usuario de un solo proceso.

**Núcleo del sistema desconoce o no interfiere en la ejecución de los hilos modo nivel usuario. Porque todas las estructuras de datos, se gestionan en el espacio de dirección de usuario de un solo proceso. Todo se ejecuta a nivel usuario. No hay cambio de contexto.**

* La planificación puede especificarse por parte de la aplicación. Una aplicación se puede beneficiar de un simple algoritmo de planificación cíclico, mientras que otra se podría beneficiar de un algoritmo de planificación basado en prioridades.
* Los ULT pueden ejecutar en cualquier sistema operativo. No se necesita ningún cambio en el nuevo núcleo para dar soporte a los ULT.

**Llamando a las librerías, del programa que se está ejecutando, se pueden crear y gestionar.**

**Desventajas:**

* En un sistema operativo típico muchas llamadas al sistema son bloqueantes. Como resultado, cuando un ULT realiza una llamada al sistema, no sólo se bloquea ese hilo, sino que se bloquean todos los hilos del proceso.

**Llamada sist: un proceso en modo usuario solicita permiso al núcleo, para poder realizar ciertas instrucciones. Cuando un proceso se está ejecutando en modo usuario, necesita mostrar un dato por pantalla (E/S), ese proceso se bloquea, son procesos bloqueantes. Cuando un hilo a nivel usuario realiza una llamada a sist. no solo bloquea a ese hilo, sino que bloquea a todos los hilos del proceso. Porque todo se ejecuta en permiso de usuario.**

**Si creamos 4 hilos a nivel usuario vamos a tener min. Uno nivel núcleo. Cuando uno se bloquee se bloquea el núcleo, por lo tanto, el resto no se puede ejecutar. Porque está bloqueando el proceso ese hilo.**

* En una estrategia pura ULT, una aplicación multihilo no puede sacar ventaja del multiproceso. El núcleo asigna el proceso a un solo procesador al mismo tiempo. Por consiguiente, en un determinado momento sólo puede ejecutar un hilo del proceso.

**A nivel núcleo los hilos son independientes de las tareas que realicen, a nivel usuario siempre van a depender al menos de un hilo a nivel núcleo.**

**Hilos Nivel Kernel - KLT->KernelLevelThreads**

* En un entorno KLT puro, el núcleo gestiona todo el trabajo de gestión de hilos.
* No hay código de gestión de hilos en la aplicación (**no a nivel usuario**), solamente una interfaz de programación de aplicación (API) para acceder a las utilidades de hilos del núcleo.

**Ventajas:**

* El núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores.
* Si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo del mismo proceso.

**Si se bloquea un hilo de ese proceso, si tenemos otro hilo ese podría ejecutarse.**

* Las rutinas del núcleo pueden ser en sí mismas multihilo.

**Desventajas:**

* La transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.

**Tiempo que se pierde entre pasar un hilo de modo usuario a modo núcleo.**

**Cuando se crea un proceso, vamos a tener 1 hilo. Lo min que vas a tener una parte ULT, KLT. (1 núcleo y 1 usuario)**

**Si esa puro LT, por un hilo de núcleo uno o mas hilos a nivel usuario. Todos dependen del hilo nivel núcleo.**

**Si es puro núcleo, se puede tener uno o más hilos niveles núcleo y uno a nivel usuario que funciona como una API, como para ir pasándole los datos.**

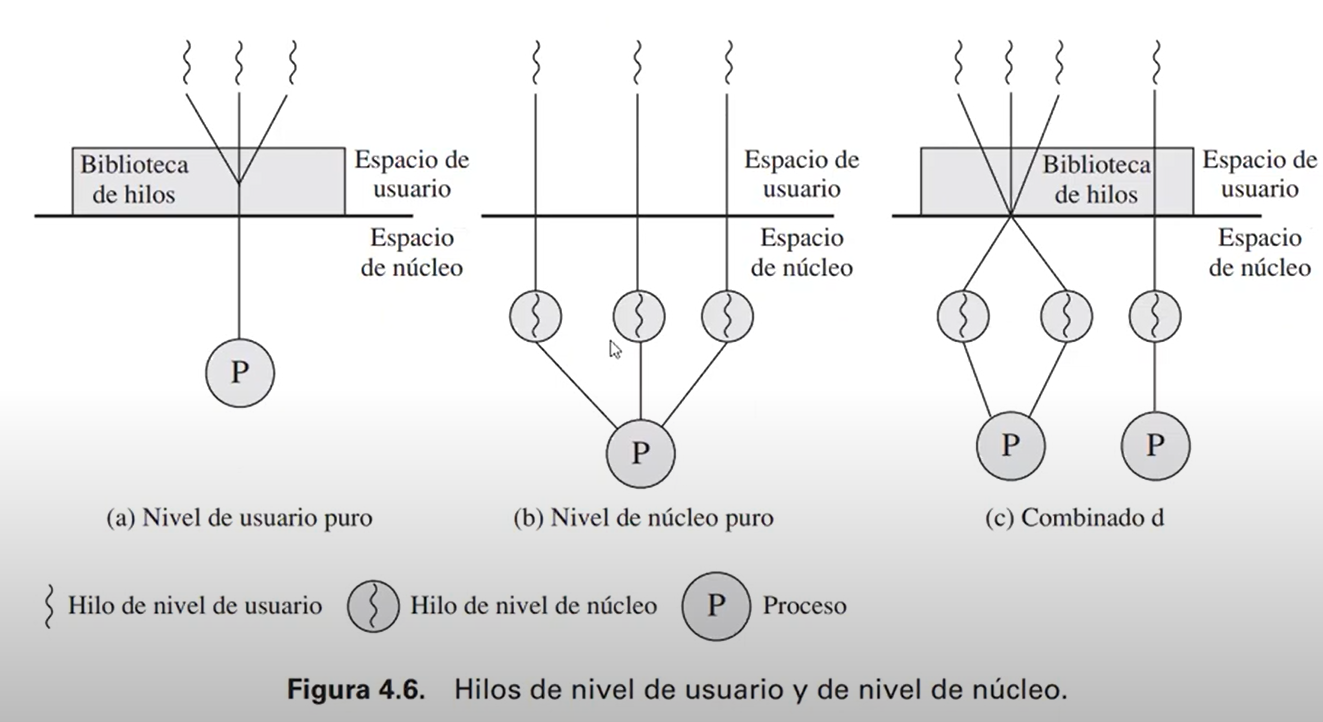
**Enfoques Combinados**

AlgunossistemasoperativosproporcionanutilidadescombinadasULT/KLT.

* La transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.
* La creación de hilos se realiza por completo en el espacio de usuario, como la mayor parte de la planificación y sincronización de hilos dentro de una aplicación.
* Los múltiples ULT de una aplicación se asocian en un número (menor o igual) de KLT.

**Ventajas:**

* En los enfoques combinados, múltiples hilos de la misma aplicación pueden ejecutar en paralelo en múltiples procesadores, y una llamada al sistema bloqueante no necesita bloquear el proceso completo.
* Si el sistema está bien diseñado, este enfoque debería combinar las ventajas de los enfoques puros ULT y KLT, minimizando las desventajas.



**Figura 1. Nivel usuario. En caso que se ejecute un hilo a nivel usuario, bloquea el proceso. Main tiene parte núcleo y parte usuario.**

**Figura 2. Nivel núcleo puro. Si se bloquea ya sea un hilo usuario o núcleo el proceso puede seguir con otro hilo.**

**Figura 3. Combinado. Combinar las ventajas de ambos niveles.**

**Librería para C**

**pthread.h**

**#include <pthread.h>**

**Creación de Hilos -Pthread**

**Int pthread\_create ( pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*func) (void \*), void \*arg)**

* Crea un hilo e inicia su ejecución.
* **Thread:** se debe pasar la dirección de una variable del tipo pthread que se usa como manejador de hilo
* **Attr:** se debe de pasar la dirección de una estructura con los atributos del hilo se puede pasar
* **Func:** función con el código de ejecución del hilo
* **Arg:** puntero al parámetro del hilo Solo se puede pasar un parámetro

**La instrucción para crear y ejecutar un hilo es pthread\_create, va a devolver un entero. Está función debe recibir siempre 4 parámetros.**

**pthread\_t \*thread - El nombre que le vamos a poner al hilo**

**const pthread\_attr\_t \*attr - Dato de tipo atributo, se le da el comportamiento al hilo. La prioridad que tenga, paramentro al hilo, planificación del hilo, si es dependiente o independiente. También se puede pasar Null.**

**void \*(\*func) (void \*) - Luego se le va a pasar la función. Tarea que a realizar por el hilo**

**void \*arg - Argumento, parámetro que queremos que utilice la función. (entero, carácter, etc)**

**Espera y Terminación**

**Int pthread\_join (pthread\_tthread, void\*\*value)**

* El hilo que invoca la función se espera hasta que el hilo cuyo manejador se especifique haya terminado
* **Thread:** manejador del hilo al que hay que esperar
* **Value:** valor de terminación del hilo.

**El primer porgrama que se ejecuta en C es el main, y detro de este se van a ir realizando todas las instrucciones, en forma secuencial de de la definición de los hilos la creación de los hilos.**

**Se crean 3 hilo cada uno de ellos realizan 3 funciones. Si una vez creados se llama a la función de exit del main, lo que va hacer es cerrar el hilo principal. Por ende, se cierran el resto de los hilos. Porque están a nivel usuario. Si se cierra el núcleo del cual dependían todos los usuarios, se cierra el resto.**

**Con esta función hace que el main espere a la finalización de todos los hilos que se hayan creado dentro del programa.**

**Int pthread\_exit (void \*\* value)**

* Permite a un proceso ligero (o hilos) planificar su ejecución, indicando el estado de terminación del mismo
* El estado de terminación no puede ser un puntero a una variable local.

**Básicamente lo que hace es finalizar el hilo que se está ejecutando.**

**Identificación de Hilos -Pthread**

**Pthread\_t pthread\_self (void)**

Devuelve el identificador del thread que ejecuta la llamada.

**Devuelve el ID del hilo.**

**Atributos de un Hilo**

* Cada hilo tiene asociados un conjunto de atributos
* Atributos representados por una variable de tipo **phtread\_attr\_t**
* Los atributos controlan si un hilo es independiente o dependiente, el tamaño de la pila privada del hilo, la localización de la pila del hilo y la política de planificación del hilo

**Int pthread\_attr\_init (pthread\_atrr\_t \*attr)**

inicia una estructura de atributos de hilo.

**Int pthread\_attr\_destroy (pthread\_atrr\_t \*attr)**

destruye una estructura de atributos de hilo

**Int pthread\_attr\_setStackSize (pthread\_atrr\_t \*attr, int stackSize)**

define el tamaño de la pila para un hilo.

**Int tpthread\_attr\_getStackSize (pthread\_atrr\_t \*attr, int \*stackSize)**

permite obtener el tamaño de la pila de un hilo