## HILOS

**Definición**

Son características que permite a una aplicación realizar varias tareas a la vez. Es una tarea que puede ser ejecutada en paralelo con otra tarea.

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

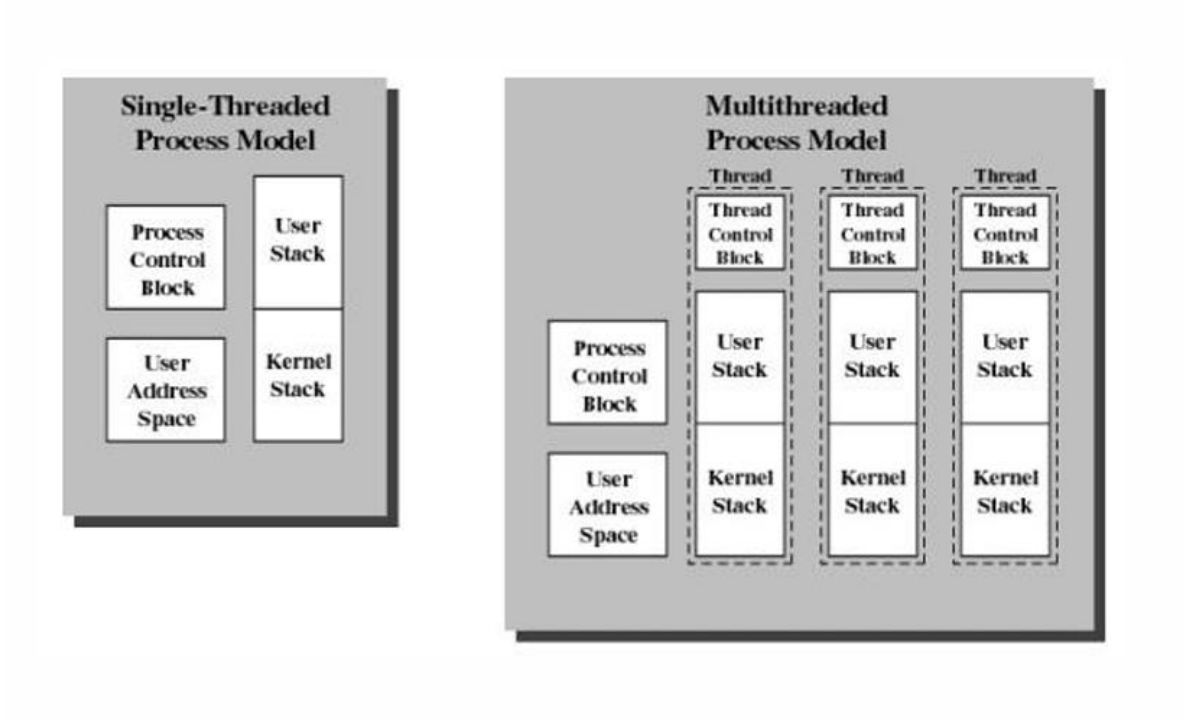
Se lo considera una unidad básica de utilización de la CPU.

Cada hilo posee: identificador del thread, controlador de programa, conjunto de registros y pila.

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

**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.
2- Lleva menos tiempo finalizar un hilo que un proceso.
3- Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
4- Los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando.



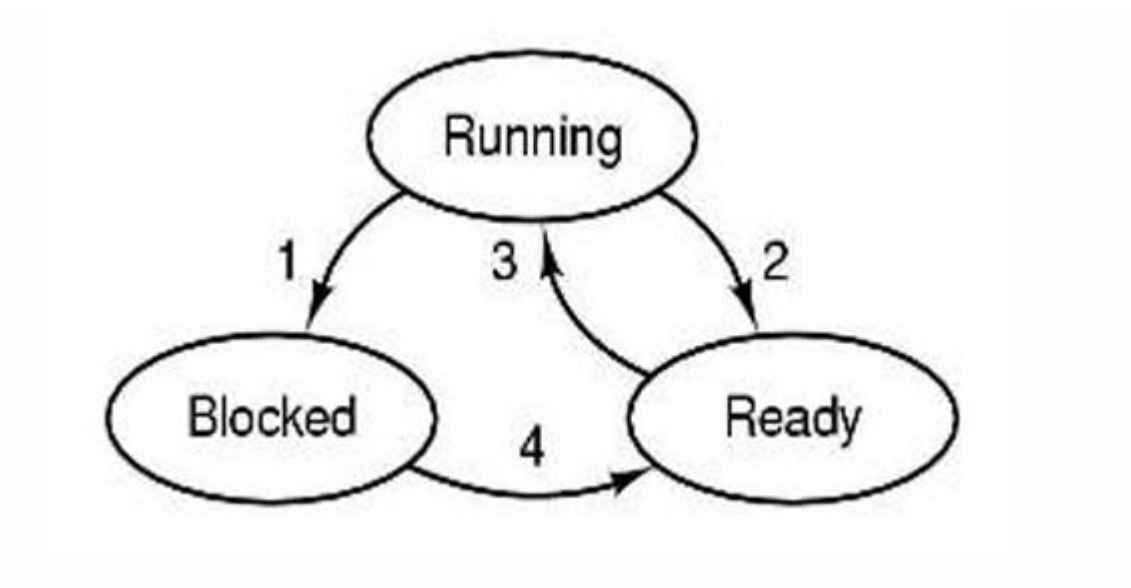
**ESTADOS DE UN HILO**

Un thread puede estar en cualquiera de estos 3 estados:

**Ejecución->** tiene en ese momento la CPU y esta activo.

**Bloqueado->** espera que algún proceso lo desbloquee.

**Listo->** esta planificado para ejecutarse y lo hace apenas llegue su turno.




**HILOS: NIVEL USUARIO**

**ULT– User Level Threads**

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

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

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.

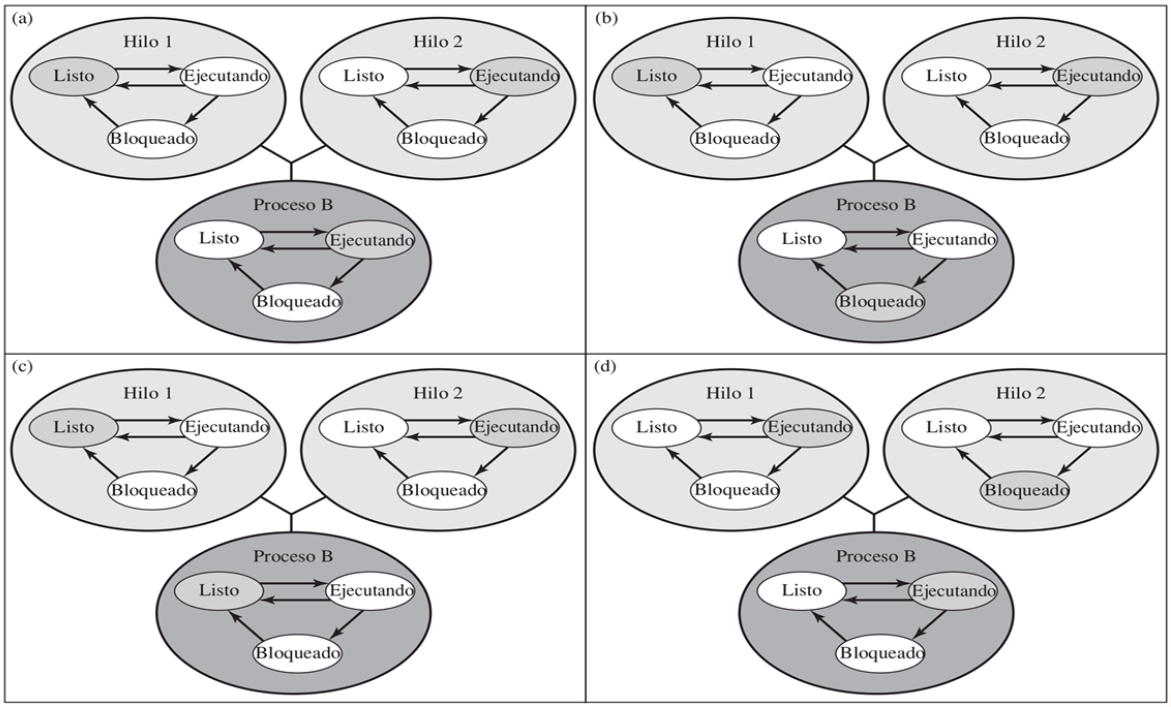
Los ULT pueden ejecutar en cualquier sistema operativo. No se necesita ningún cambio en el nuevo núcleo para dar soporte a los ULT.

**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.

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.

Ejemplo de las relaciones entre los estados de los hilos de nivel usuario y los estados de proceso.




**HILOS: NIVEL KERNEL**

**KLT-> Kernel Level Threads**

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, 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.

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.

**ENFOQUES COMBINADOS**

Algunos sistemas operativos proporcionan utilidades combinadas **ULT/KLT.**

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

En un sistema combinado, 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.

**CREACION 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 dehilo.

**Attr:** se debe de pasar la dirección de una estructura con los atributos del hilo se puede pasar NULL para usar atributos por defecto.

**Func:** función con el código de ejecución del hilo.

**Arg:** puntero al parámetro del hilo. Solo se puede pasar un parámetro.

**ESPERA Y TERMINACION**

**Int pthread_join(pthread_t thread, 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.

**Int pthread_exit(void **value)**

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

**IDENTIFICACIÓN DE HILOS PTHREAD**

**Pthread_t pthread_self(void)->** devuelve el identificador del thread que ejecuta la llamada.

**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 estruct de atrubitos de hilo.

**Int pthread_attr_setStackSize (pthread_atrr_t *attr, int stackSize)->** define el tamaño de la pila para un hilo.

I**nt pthread_attr_getStackSize (pthread_atrr_t *attr, int *stackSize)->** permite obtener el tamaño de la pila de un hilo