¿Qué son los *hilos*?

En [sistemas operativos](https://es.wikipedia.org/wiki/Sistema_operativo), un **hilo de ejecución**, **hebra** o **subproceso** es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo con otra tarea.

La creación de un nuevo hilo es una característica que permite a una [aplicación](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica) realizar varias tareas a la vez ([concurrentemente](https://es.wikipedia.org/wiki/Concurrencia)). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un [proceso](https://es.wikipedia.org/wiki/Proceso_(inform%C3%A1tica)). El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

Lo que es propio de cada hilo es el [contador de programa](https://es.wikipedia.org/wiki/Contador_de_programa), la [pila de ejecución](https://es.wikipedia.org/wiki/Pila_de_ejecuci%C3%B3n) y el estado de la [CPU](https://es.wikipedia.org/wiki/CPU) (incluyendo el valor de los [registros](https://es.wikipedia.org/wiki/Registro_de_pila)).

El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

Algunos [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) tienen características de diseño expresamente creadas para permitir a los [programadores](https://es.wikipedia.org/wiki/Programador) lidiar con hilos de ejecución (como [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) o [Delphi](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Delphi" \o "Lenguaje de programación Delphi)). Otros (la mayoría) desconocen la existencia de hilos de ejecución y éstos deben ser creados mediante llamadas de biblioteca especiales que dependen del sistema operativo en el que estos lenguajes están siendo utilizados (como es el caso del [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) y del [C++](https://es.wikipedia.org/wiki/C%2B%2B)).

Diferencia entre *Hilos* y *Procesos*

Los hilos se distinguen de los tradicionales procesos en que los procesos son independientes, llevan bastante información de estados, e interactúan sólo a través de [mecanismos de comunicación](https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_entre_procesos) dados por el sistema. Por otra parte, muchos hilos generalmente comparten otros recursos de forma directa. En muchos de los [sistemas operativos](https://es.wikipedia.org/wiki/Sistema_operativo) que dan facilidades a los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen. Al cambiar de un proceso a otro el sistema operativo (mediante el dispatcher) genera lo que se conoce como overhead, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (context switch), en este caso pasar del estado de ejecución (running) al estado de espera (waiting) y colocar el nuevo proceso en ejecución. En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.

API de *Hilos*

La función principal es obviamente la que permite crear un thread. Esta función es pthread\_create:

int pthread\_create(pthread\_t \* thread

, pthread\_attr\_t \* attr

, void \*(\*funcion)(void \*)

, void \*arg);

Los parámetros son los siguientes:

**thread**

Un puntero en donde la función guardará el identificador del nuevo thread creado.

**attr**

Atributos del thread, puede ser NULL.

**funcion**

Acá empieza la ejecución del nuevo thread. Es un puntero a una función declarada de esta manera: void mi\_thread(void \*arg);.

**arg**

Parámetro a pasar a la función del nuevo thread

|  |  |
| --- | --- |
| **Tipo de dato** | **Descripción** |
| pthread\_attr\_t | Atributo de hilo |
| pthread\_mutexattr\_t | Atributo de mutex |
| pthread\_condattr\_t | Atributo de variable de condición |
| pthread\_mutex\_t | Mutex (bloqueo con exclusión mutua) |
| pthread\_cond\_t | Variable de condición |
| pthread\_t | Hilo (identificador de hilo o ID) |
| pthread\_once\_t | Ejecución una sola vez |
| pthread\_key\_t | Clave sobre datos específicos de hilo |

**Gestión de hilos**

Las funciones básicas para la gestión de hilos son las siguientes:

|  |  |
| --- | --- |
| **Función** | **Descripción** |
| pthread\_create | Crea un hilo |
| pthread\_equal | Verifica la igualdad de dos identificadores de hilo |
| pthread\_exit | Termina el hilo que realiza la llamada |
| pthread\_join | Espera por el termino de un hilo especifico |
| pthread\_self | Regresa el ID del hilo que realiza la llamada |
| pthread\_detach | Configura la liberación de recursos cuando termina (hilo independiente) |
| pthread\_getschedparam | Obtiene la política de planificación y parámetros de un hilo específico |
| pthread\_setschedparam | Establece la política de planificación y parámetros de un hilo específico |
| pthread\_kill | Envía una señal de determinada a un hilo específico |
| pthread\_cancel | Permite a un hilo cancelar otro hilo del mismo proceso |

**Creando un hilo (pthread\_create)**

Esta función automáticamente pone en ejecución el hilo que crea. La sintaxis es la siguiente:

int pthread\_create( pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void\*), void \*arg);

Donde:

* El parámetro thread apunta al ID del hilo recientemente creado.
* El parámetro attr representa un objeto atributo que encapsula los atributos de un hilo. Si attr es NULL, el hilo nuevo tendrá los atributos asignados por defecto.
* El tercer parámetro, start\_routine, es el nombre de la función que es invocada por el hilo cuando comienza su ejecución.
* El parámetro arg especifica el parámetro que recibe la función start\_routine

Si pthread\_create se ejecuta satisfactoriamente retorna 0. Si la rutina no se ejecuta satisfactoriamente retorna un código de error diferente de cero. A continuación se muestra una tabla con los errores que puede generar una invocación a pthread.

|  |  |
| --- | --- |
| **Error** | **Causa** |
| EAGAIN | El sistema no posee los recursos necesarios para crear el nuevo hilo, o se excede el límite total del número de hilos permitidos por el sistema. |
| EINVAL | El parámetro attr es inválido |
| EPERM | No se tienen los permisos suficientes para cambiar la política de planificación o cualquier otro parámetro especificado en attr |

* **Sobre el objeto atributo**

Los atributos se almacenan en un objeto atributo de tipo pthread\_attr\_t.

Para la creación y el inicio de un objeto atributo (utilizado en la creación de un proceso ligero) se utiliza la siguiente función:

int pthread\_attr\_init(pthread\_attr\_t \*attr);

Para la destrucción del objeto de tipo atributo se utiliza:

int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

El servicio para el establecimiento (set) del atributo correspondiente al estado de terminación es:

int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

El valor del argumento detachstate puede ser:

* PTHREAD\_CREATE\_DETACHED: El proceso ligero que se cree con este estado de terminación se considerará independiente y liberará sus recursos cuando finalice su ejecución.
* PTHREAD\_CREATE\_JOINABLE: El proceso ligero que se cree con este estado de terminación se considerará como no independiente y no liberará sus recursos cuando finalice su ejecución. En este caso, es necesario que otro proceso espere por su finalización utilizando pthread\_join.

El servicio para la obtención (get) del atributo correspondiente al estado de terminación es el siguiente:

int pthread\_attr\_getdetachstate(pthread\_attr\_t \*attr, int \*detachstate);

Cada hilo tiene una pila cuyo tamaño se puede establecer en el momento de la creación, dándole al atributo que se va a utilizar en la creación el valor adecuado. Esto se hace mediante el servicio:

int pthread\_attr\_setstacksize (pthread\_attr\_t \*attr, int stacksize);

El servicio para obtener el tamaño de la pila es el siguiente:

int pthread\_attr\_getstacksize (pthread\_attr\_t \*attr, int \*stacksize);

**Comparando IDs (pthread\_equal)**

El ID de un hilo es  dependiente del SO y puede ser una estructura, por esta razón se deberá utilizar una función para comparar la igualdad de los IDs de los hilos

int pthread\_equal(pthread\_t t1, pthread\_t t2);

Donde t1 y t2 son IDs de los hilos que van a ser comparados.

Si t1 es igual a t2, pthread\_equal retorna un valor diferente de cero. Si los IDs de los hilos son diferentes, pthread\_equal retorna 0.

**Salir (pthread\_exit)**

Un hilo puede terminar de tres maneras sin terminar el proceso: retornando de su rutina de inicio, cancelado por otro hilo del mismo proceso, o llamando pthread\_exit.

void pthread\_exit(void \*value\_ptr);

El valor de value\_ptr debe apuntar a datos que existan inclusive después que el hilo termine.

**Unir (pthread\_join)**

La función pthread\_join suspende la ejecución del hilo que la invoca hasta que el hilo objetivo, especificado por el primer parámetro, termine.

int pthread\_join(pthread\_t, void \*\*value\_ptr);

El parámetro **valor\_ptr** es un apuntador al valor de retorno que el hilo objetivo pasa a la función pthread\_exit o a return. Si value\_ptr es NULL, el hilo que invoca join no recibe el valor de retorno del hilo objetivo.

Si la función termina exitosamente retorna 0. Si no termina exitosamente retorna un valor de error diferente de cero.

|  |  |
| --- | --- |
| **Error** | **Causa** |
| EINVAL | El hilo no corresponde a un hilo disponible para la unión |
| ESRCH | No existe un hilo identificado por ID |

**Referencias a hilos por su id (pthread\_self)**

Cualquier hilo puede conocer su ID invocando a:

pthread\_t pthread\_self(void);

Esta función retorna el ID del hilo que la invoca.

**Separar (pthread\_detach)**

Cuando un hilo termina no libera sus recursos al menos que el hilo este separado. La función pthread\_detach modifica las opciones internas del hilo para especificar que el espacio utilizado para el almacenamiento del mismo puede ser reclamado cuando culmine. Los hilos ya separados no reportan su estado cuando culminan.

int pthread\_detach( pthread\_t thread );

La función pthread\_detach recibe un solo parámetro, thread, que es el ID del hilo a ser separado.

Si pthread\_detach culmina exitosamente retorna uno. Si no culmina exitosamente retorna un valor diferente de cero. La siguiente tabla muestra los códigos de error de pthread\_detach.

|  |  |
| --- | --- |
| Error | Causa |
| EINVAL | El hilo no corresponde a un hilo disponible para la unión |
| ESRCH | No existe un hilo identificado por ID |

**Cancelar (pthread\_cancel)**

Un hilo invoca pthread\_cancel para solicitar que otro hilo sea cancelado. El resultado de la invocación es determinado por el tipo del hilo objetivo y su estado de cancelación.

int pthread\_cancel(pthread\_t thread);

El parámetro de pthread\_cancel es el ID del hilo objetivo a ser cancelado.

Si pthread\_cancel se ejecuta exitosamente retorna 0. Sino retorna un valor diferente de cero.

**Mutex para problemas de concurrencia**

La concurrencia es la propiedad de los sistemas que permiten que múltiples [procesos](http://www.alegsa.com.ar/Dic/proceso.php) sean [ejecutados](http://www.alegsa.com.ar/Dic/ejecutar.php) al mismo tiempo, y que potencialmente puedan interactuar entre sí.

Trabajar con hilos es trabajar nativamente con programas concurrentes, uno de los mayores problemas con los que nos podremos encontrar, y que es tácito en la concurrencia, es el acceso a variables y/o estructuras compartidas o globales, es decir usar variables que son modificadas por otros hilos.

Existen regiones críticas (RC) que son parte de código susceptible de verse afectada por la concurrencia.

Para solucionar el problema los hilos POSIX nos ofrecen los semáforos binarios, semáforos mutex o simplemente mutexs. Un semáforo binario es una estructura de datos que actúa como un semáforo porque puede tener dos estados: abierto o cerrado. Cuando el semáforo está abierto, al primer hilo que pide un bloqueo se le asigna ese bloqueo y no se deja pasar a nadie más por el semáforo. Mientras que si el semáforo está cerrado, porque algún hilo ya tiene el bloqueo, el thread que lo pidió parará su ejecución hasta que no sea liberado el susodicho bloqueo. Solo puede haber un solo hilo teniendo el bloqueo del semáforo, mientras que puede haber más de un hilo esperando para entrar en la RC, situados en la cola de espera del semáforo. Es decir, los threads se excluyen mutuamente (de ahí lo de mutex para el nombre) el uno al otro para entrar.

Las principales funciones para el uso de mutex son las siguientes:

|  |  |
| --- | --- |
| **Función** | **Descripción** |
| pthread\_mutex\_init | Inicializa un mutex con los atributos especificados |
| pthread\_mutex\_destroy | Destruye el mutex especificado |
| pthread\_mutex\_lock | Permite solicitar acceso al mutex, el hilo se bloquea hasta su obtención |
| pthread\_mutex\_trylock | Permite solicitar acceso al mutex,  el hilo retorna inmediatamente. El valor retornado indica si otro hilo lo tiene. |
| pthread\_mutex\_unlock | Permute liberar un mutex. |
| pthread\_mutex\_getprioceiling | Obtiene el valor de prioridad límite (prioceiling) del mutex especificado. |
| pthread\_mutex\_setprioceiling | Establece el valor de prioridad límite y devuelve el anterior valor (old-ceiling) en el mutex especificado. |

**Inicialización de mutex**

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr)

**mutex**: Es un puntero a un parámetro del tipo pthread\_mutex\_t, que es el tipo de datos que usa la librería Pthreads para controlar los mutex.

**attr**: Es un puntero a una estructura del tipo pthread\_mutexattr\_t y sirve para definir qué tipo de mutex queremos: normal, recursivo o errorcheck. Si este valor es NULL (recomendado), la librería le asignará un valor por defecto.

La función devuelve 0 si se pudo crear el mutex o -1 si hubo algún error. Esta función debe ser llamada antes de usar cualquiera de las funciones que trabajan con mutex.

**Petición de bloqueo**

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex)

**mutex:** Es un puntero al mutex sobre el cual queremos pedir el bloqueo o sobre el que nos bloquearemos en caso de que ya haya alguien dentro de la RC.

Esta función pide el bloqueo para entrar en una RC. Si queremos implementar una RC, todos los thread tendrán que pedir el bloqueo sobre el mismo semáforo. Como resultado, devuelve 0 si no hubo error, o diferente de 0 si lo hubo.

**Liberación de bloqueo**

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex)

**mutex**: Es el semáforo donde tenemos el bloqueo y queremos liberarlo.

Esta función libera el bloqueo que tuviéramos sobre un semáforo. Retorna 0 como resultado si no hubo error o diferente de 0 si lo hubo.

**Destrucción de mutex**

int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex)

**mutex:** El mutex que queremos destruir.

Esta función le dice a la librería que el mutex que el estamos indicando no lo vamos a usar más, y que puede liberar toda la memoria ocupada en sus estructuras internas por ese mutex. La función devuelve 0 si no hubo error, o distinto de 0 si lo hubo.

**Programas con uso de hilos**

Como en nivel y la rapidez de procesamiento de las computadoras actuales es más rápido que en épocas anteriores, la ejecución de un hilo es difícil de visualizar, pues prácticamente se realiza de una forma transparente para el usuario.  
En esta parte se explicaran ciertos programas que hacen uso de hilos, cada uno realizando cierta acción para ver de una manera más fácil la ejecución de los hilos.

**Creación de hilos**

Este programa es muy sencillo ya que realiza la creación de dos hilos, los cuales imprimen la dirección en la cual se encuentran creado el hilo.

Lo primero que se realiza es llamar a la librería en donde se encuentran ubicadas las funciones para el uso de los hilos.

#include <pthread.h>

#include <stdio.h>

Lo siguiente es crear una función que será llamada posteriormente al crearse el hilo, esta función lo único que realiza es la impresión de la dirección del hilo sin recibir ningún parámetro. Para obtener la dirección del hilo se utiliza la función pthread\_self()

void func(void){

printf("Thread %x \n", pthread\_self()); //Id del hilo creado

pthread\_exit(NULL);

}

Después se crea la función main, en la cual se crean los hilos a utilizar y las variables donde estarán las direcciones de cada hilo.

int main()

{

pthread\_t th1, th2; //Direcciones donde estarán los hilos

//Se crean dos hilos sin parametros y con atributos por defectos

pthread\_create(&th1, NULL, (void \*)func, NULL);

pthread\_create(&th2, NULL, (void \*)func, NULL);

printf("El proceso ligero principal continua ejecutando\n");

//Se espera a que terminen los hilos

pthread\_join(th1, NULL);

pthread\_join(th2, NULL);

printf("Se terminaron de ejecutar los hilos\n");

return(0);

}

Lo que hace pthread\_create(&th1, NULL, (void \*)func, NULL); es crear un hilo tomando cierto parámetros para inicializarlo.

&th1 🡪 Contiene la dirección del hilo creado

NULL 🡪 Inicializa por defecto los atributos del hilo

(void \*)func 🡪 Llama a la función a ejecutar, necesita un casteo porque recibe como parámetro un puntero

NULL 🡪 Argumentos de la función, en este caso no se necesita ninguno.

pthread\_join(th1, NULL); Espera a la terminación del hilo

th1 🡪 Identificador del hilo, espera a que termine de ejecutarse

NULL 🡪 Regresa un valor cuando ha terminado el hilo.

Ejecutando este programa lo que se obtendría seria:

