

Hebras

Gustavo Romero

Arquitectura y Tecnología de Computadores

14 de febrero de 2014

Índice

1 Introducción

2 Tipos

- Hebras tipo usuario
- Hebras tipo núcleo
- Hebras híbridas

3 Representación

4 Ejemplos

- Windows
- Linux
- Bibliotecas

Lecturas recomendadas

- Tanuenbaum Sistemas Operativos Modernos (2.2)
- Silberschatz Fundamentos de Sistemas Operativos (4)
- Stallings Sistemas Operativos (4)

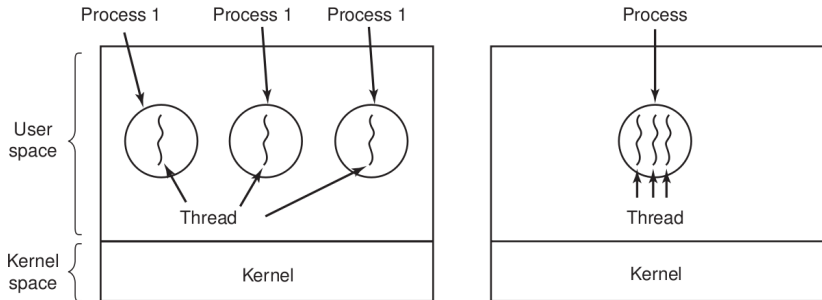
Introducción

Definición

- Proceso en un SO “*tradicional*” = espacio de direcciones + hebra de control.
- Proceso = contenedor de recursos + hebra de ejecución.
- Una hebra se ejecuta **dentro** de un proceso.
- Proceso y hebra son conceptos diferentes y suelen ser tratados por separado.
 - La función de un proceso es **agrupar recursos**.
 - Las hebras son **entidades planificables** para su ejecución sobre un procesador.
- Las hebras añaden a los procesos la capacidad de ejecutar varios conjuntos de instrucciones de forma concurrente dentro de un mismo entorno de procesamiento.
- Ejecutar varias hebras en paralelo es parecido a ejecutar varios procesos salvo por que comparten su recursos.
- A veces son llamados **procesos ligeros** o subprocesos.

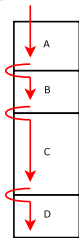
Comparación proceso/hebra

- izquierda: 3 procesos, cada uno de ellos con una hebra.
- derecha: 1 proceso con 3 hebras.

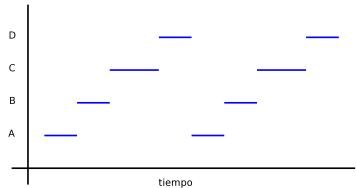
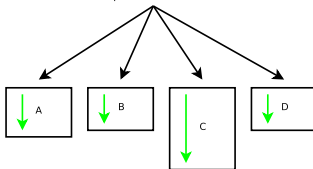


- En ambos casos tenemos 3 hebras.
- En un caso necesitamos 3 PCB y en el otro 1.

- un puntero de instrucción



cuatro punteros de instrucción



Ventajas de la multihebra

- Mejora de rendimiento:
 - Lleva 10 veces menos tiempo crear una nueva hebra que un proceso.
 - Se tarda mucho menos en finalizar un hebra que un proceso.
 - Se tarda mucho menos en conmutar entre dos hebras dentro de un mismo proceso que entre dos procesos.
 - Las comunicaciones entre hebras son potencialmente más eficientes porque (según el tipo) no requieren la intervención del núcleo.
- Ejemplos de uso:
 - Trabajo simultáneo en primer y segundo plano, ej: procesador de texto, servidor de ficheros,...
 - Procesamiento asíncrono, ej: copia de seguridad.
 - Velocidad de ejecución: descomposición de problemas y procesamiento paralelo, ej: solapamiento de obtención de datos y cálculo.
 - Modularidad de programas: fácil descomposición de tareas complejas en otras más simples.

Independencia/Cooperación

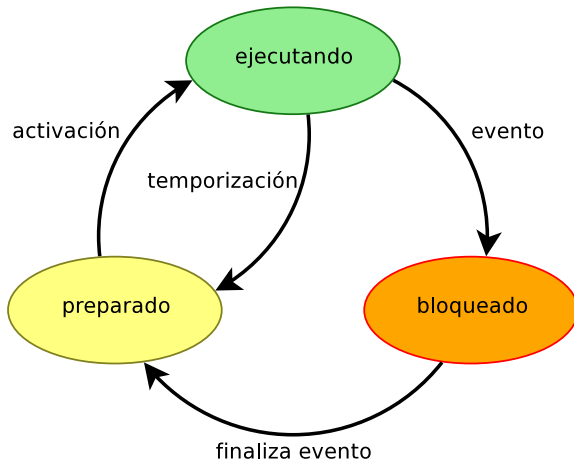
- Las diferentes hebras de un proceso **no** son **independientes** como las que se encuentran en diferentes procesos.
- Todas las hebras **comparten** los mismos **recursos**: espacio de direcciones, variables globales, ficheros abiertos,...
- Una hebra puede **alterar** cualquier parte de otra con la que comparta proceso.
- **No** hay **protección** entre hebras porque...
 - **es imposible**: comparten recursos.
 - **no es necesario**: diferentes procesos pueden pertenecer a diferentes usuarios, en cambio, todas las hebras de un proceso pertenecen al mismo.

Recursos necesarios

por proceso	por hebra
espacio de direcciones variables globales ficheros abiertos procesos hijos alarmas pendientes señales y manejadores de señales información contable ...	identificador registros pila estado datos privados

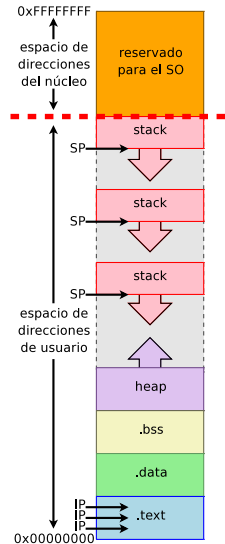
Estado de un hebra

- Al igual que los procesos, una hebra puede pasar por diferentes estados a lo largo de su ejecución.



Espacio de direcciones

- Cada hebra necesita su **propia pila**.
- Una pila contiene un **marco** de pila **por** cada **procedimiento** llamado pero que no ha retornado todavía.
- Si en un hebra ejecutamos el procedimiento X, este llama al Y, y este llama Z, durante la ejecución de Z la pila contendrá los marcos de X, Y y Z.



Gestión de hebras

- Para gestionar las hebras, al igual que sucede con los procesos, necesitamos una serie de **llamadas al sistema**.
- Los procesos multihebra suelen comenzar su ejecución con una **única hebra** y esta puede llamar a `thread_create()` si necesita crear otras.
- Las hebras pueden mantener una relación jerárquica o ser creadas todas iguales (más habitual).
- Cada hebra posee un identificador único.
- Es necesaria un llamada al sistema para terminar con una hebra: `thread_exit()`.
- Al igual que los procesos, podemos hacer que las hebra sincronicen su funcionamiento: `thread_join()`.

Gestión de hebras

- Otra llamada habitual es `thread_yield()`, que permite a una hebra ceder voluntariamente el procesador.
 - muy importante porque puede que la interrupción de reloj no las afecte y no se pueda imponer el tiempo compartido.
 - las hebras deben cooperar.
- Las hebras introducen nuevos problemas en el sistema:
 - `fork()`: si un padre tiene varias hebras, ¿debería el hijo tenerlas también?
 - Si el proceso padre tiene una hebra bloqueada en espera de una entrada desde el teclado, ¿debería tenerla también el hijo?
 - Este mismo problema puede darse con ficheros, conexiones de red, gestión de memoria,...
- La mayoría de estos problemas pueden resolverse con una cuidadosa coordinación entre las hebras.

Motivos para el uso de hebras

- **Compartición** de recursos, ej: servidor web.
- **Facilidad** (economía) de creación y destrucción.
- Mejora del **rendimiento**:
 - gran mejora en los limitados por E/S.
 - pequeña mejora en los limitados por procesador.
- En sistemas con múltiples procesadores es posible un verdadero **paralelismo**.
- **Simplicidad** de programación (a veces), ej: procesador de texto.

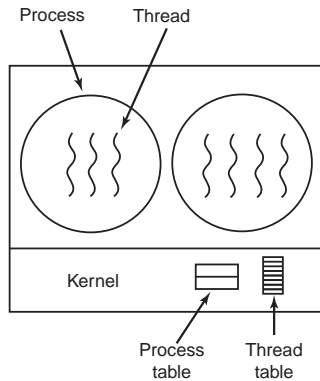
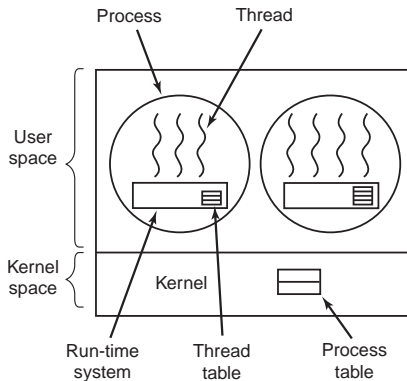
Ejemplos de uso de hebras

- Procesador de textos: utilizará una hebra por cada tarea que realiza.
 - Interacción con el usuario.
 - Corrector ortográfico/gramatical.
 - Guardar automáticamente y/o en segundo plano.
 - Mostrar resultado final en pantalla (WYSIWYG).
- Hoja de cálculo:
 - Interacción con el usuario.
 - Actualización en segundo plano.
- Servidor web: dos tipos de hebras.
 - Interacción con los clientes (navegadores).
 - Gestión del caché de páginas.
- Navegador: varios tipos de hebras.
 - Interacción con el usuario.
 - Interacción con los servidores (web, ftp, ...).
 - Dibujo de la página (1 hebra por pestaña).

Tipos

Tipos de hebras

- Hebras tipo usuario.
- Hebras tipo núcleo.
- Implementaciones híbridas.



Hebras tipo usuario

- El sistema de hebras se coloca por completo en el espacio de usuario.
- El núcleo no sabe nada de ellas y por lo que a él respecta se dedica a planificar procesos ordinarios.
- Todos los SO comenzaron perteneciendo a este tipo.
 - la mayoría ha evolucionado hacia el uso de hebras tipo núcleo.
- Cada proceso necesita una tabla de hebras.
 - Análoga a las tablas de procesos.
 - Gestionada por el sistema de ejecución de hebras.

Ventajas de las hebras tipo usuario

- Pueden implementarse sobre **cualquier SO**.
- La **conmutación** entre hebras es varios órdenes de magnitud más **rápida** que la de procesos.
- Cuando una hebra termina de ejecutarse (`thread_exit()` / `thread_yield()`) puede cambiarse a otra **sin involucrar al núcleo** con lo que ahorraremos uno o dos cambios de proceso y modo.
- Cada proceso puede utilizar un algoritmo de **planificación personalizado**.

operación	hebras usuario	procesos
proceso nulo	34	11300
signal/wait	37	1840

Inconvenientes de las hebras usuario

- **Llamadas al sistema bloqueantes:** detener una hebra implica detener el proceso completo, ej: lectura de teclado, fallo de página. Soluciones:
 - modificar el SO para hacer sus llamadas no bloqueantes \Rightarrow no queremos/podemos hacerlo.
 - comprobar si las llamadas bloqueantes tendrán éxito antes de efectuarlas, ej: `select/read` \Rightarrow método engorroso.
- Como no existe interrupción de reloj no podremos ejecutar una hebra hasta que otra deje libre el procesador **voluntariamente** (cooperación obligatoria).
 - solución: solicitar interrupción periódica \Rightarrow pérdida de rendimiento.
- Las hebras mejoran el rendimiento de tareas que se bloquean a menudo, pero una vez en modo núcleo es casi igual de costoso cambiar de proceso que volver al proceso original.
- En sistemas multiprocesador no podemos asignar más de un procesador a cada proceso.

Hebras tipo núcleo

- El núcleo conoce la existencia de las hebras y se encarga de gestionarlas.
- No se necesita un sistema de gestión de hebras ni tablas de hebras en el interior de cada proceso.
- En el interior del núcleo ya disponemos de las tablas de procesos, las tablas de hebras en realidad son parte de ellas (replicando ciertas partes) \Rightarrow algunos sistemas (Linux) no distinguen entre procesos y hebras.
- La **gestión** de hebras se hace a través de **llamadas al sistema** \Rightarrow mayor coste de operación.

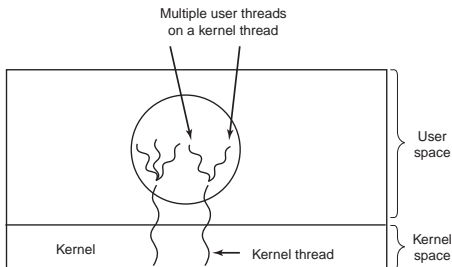
Ventajas e inconvenientes de las hebras del núcleo

- Ventajas:
 - No requieren llamadas al sistema **no bloqueantes**.
 - Si una hebra se bloquea o excede su tiempo de ejecución otra puede ser ejecutada.
 - En sistemas multiprocesador tantas hebras de un mismo proceso como procesadores existan pueden ejecutarse en **paralelo**.
- Inconvenientes:
 - **Coste** de operación mucho mayor que hebras tipo usuario.
 - Se tiende al **reciclaje** de hebras: finalizado \Rightarrow nuevo.

operación	hebras usuario	hebras núcleo	procesos
proceso nulo	34	948	11300
signal/wait	37	441	1840

Hebras híbridas

- Intentan reunir las ventajas de los tipos anteriores a la vez que evitar sus inconvenientes.
- Método:
 - Es necesario implementar hebras tipo núcleo en el SO.
 - Utiliza hebras tipo usuario pero multiplexadas sobre hebras tipo núcleo.
 - Cada hebra del núcleo ejecuta un conjunto de hebras de usuario que se turnan en su utilización.



Activaciones del planificador

- Método propuesto por Thomas E. Anderson (1992).
- El núcleo asigna cierto número de **procesadores virtuales** o **procesos ligeros** a cada proceso.
 - Inicialmente suele asignarse un procesador virtual.
 - Cada proceso puede solicitar y devolver procesadores virtuales.
 - El núcleo puede conceder y retirar procesadores virtuales.
- Cuando el núcleo detecta que una hebra se bloquea avisa al sistema de gestión de hebras en espacio de usuario mediante una llamada directa o suprrallamada (*"upcall"*).
- Una vez avisado un planificador a nivel usuario puede seleccionar otra hebra para ejecutar.
- Cuando la hebra se desbloquea de nuevo el núcleo avisa al sistema de gestión de hebras en espacio de usuario.

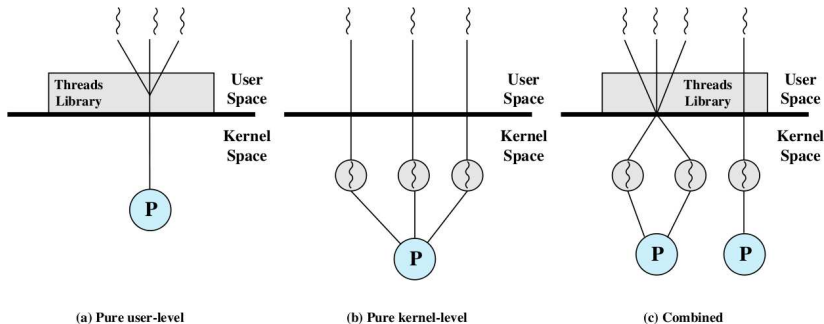
Ventajas e inconvenientes del enfoque híbrido

- Ventajas:
 - Dos formas de manejar las llamadas al sistema bloqueantes:
 - Una llamada bloqueante en una hebra de usuario sólo bloquea su correspondiente hebra del núcleo hasta que finalice la causa del bloqueo mientras que el resto de hebras del proceso continúan ejecutándose.
 - La hebra del núcleo almacena la causa del bloqueo y avisa al planificador a nivel usuario para que cambie a otra hebra de usuario hasta que finalice la causa del bloqueo.
- Inconvenientes:
 - El SO debe disponer de hebras tipo núcleo.
 - Hay dos niveles de planificación:
 - El núcleo planifica las hebras del núcleo.
 - Las hebras de usuario son planificadas por una biblioteca.
 - Ambos planificadores deben cooperar.
 - Problema: las llamadas directas violan los principios de los sistemas por capas.

En resumen...

- Hebras tipo usuario:
 - Rápidas y fáciles de implementar.
 - No planificables por el núcleo.
 - Se bloquean con E/S.
 - Sólo pueden usar un procesador.
 - Requieren cooperación entre hebras.
- Hebras tipo núcleo:
 - Involucran al SO, fraccionamiento del tiempo.
 - Operaciones de cambio y sincronización más costosas.
 - No se bloquean con E/S.
 - Pueden utilizar varios procesadores.
- Hebras híbridas:
 - Lo mejor de cada tipo.
 - Requieren balanceo de carga: reparto de hebras de usuario sobre hebras del núcleo para repartir el trabajo entre los procesadores.

Comparativa entre hebras y procesos



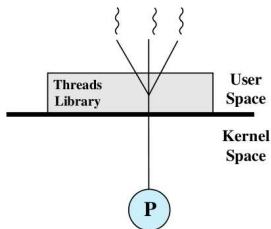
{ User-level thread
 () Kernel-level thread
 (P) Process

operación\tipo	hebras usuario	hebras híbridas	hebras núcleo	procesos
proceso nulo	34	37	948	11300
signal/wait	37	42	441	1840

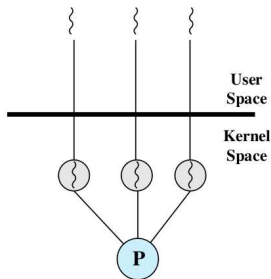
Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, Henry M. Levy.
 Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism.

ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992. (Tiempos en μs)

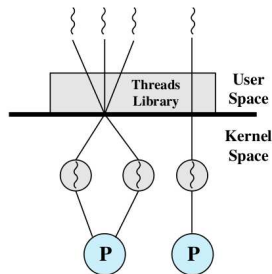
Relación entre tipos de hebras



(a) Pure user-level



(b) Pure kernel-level



(c) Combined



modelo	nº hebras (u/n)	ejemplos
muchos a uno	N:1	GNU Portable Threads, Green Threads (Solaris)
uno a uno	1:1	Linux, MacOS X, Solaris (>8), Windows (>NT)
muchos a muchos	N:M	Solaris(<9), Fiber (Windows)
dos niveles	N:M + 1:1	HP-UX, IRIX, Solaris (<9), Tru64 UNIX

Representación

Representación de hebras

- Para poder gestionar hebras necesitamos estructuras de datos que las representen \Rightarrow **bloque de control de hebra (TCB)**.

TCB mínimo
identificador de hebra (TID)
puntero de instrucción (IP)
puntero de pila (SP)
estado (flags)

Atributos de una hebra

- Identificador de hebra (TID).
- Contexto:
 - Registros de usuario.
- Planificación.
- Pila.
- Recursos privados (opcional):
 - Datos privados.
- Otros:
 - Eventos relacionados: esperando E/S.
 - Prioridades.
 - Comunicación entre procesos.
 - Información de mapeado.
 - Propiedad y utilización de recursos.

Implementación de los bloques de control de hebra

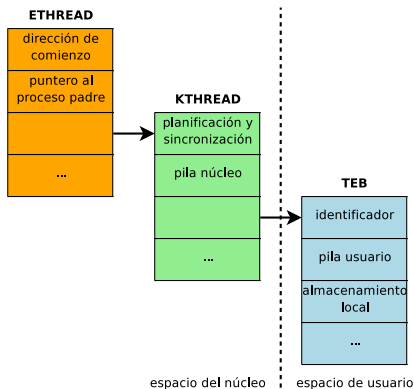
- ¿Dónde colocar los TCBs?
 - No colocarlos aleatoriamente.
 - Tenga en cuenta la TLB y la caché.
 - Diseñarlos con sumo cuidado (tanto o más que los PCBs).
- Posibles estructuras de datos:
 - Contiguas:
 - vector
 - vector de punteros.
 - No contiguas:
 - lista.
- Posibles localizaciones de los TCBs.
 - Espacio de usuario.
 - Espacio del núcleo.
 - Repartido entre ambos: $TCB = UTCB + KTCB$.

Ejemplos

Windows (95/98/NT/2000/XP)

- Implementación de la API Win32.
- Cada proceso contiene una o más hebras.
- Emplea hebras tipo usuario y núcleo con relación 1:1.
 - La biblioteca Fiber añade relaciones N:M entre hebras (opcional).
- Componentes de una hebra: identificador, conjunto de registros, pilas de usuario y núcleo, área de almacenamiento privada

- Localización de las estructuras de datos de las hebras:



Linux

- No distingue entre procesos y hebras: **tarea** (*task*).
- Utiliza la llamada al sistema `clone()` para conseguir un efecto parecido a `fork()` pero permite limitar los recursos compartidos.
- Estructura de datos para representar tareas: `task_struct`.
 - Formada por punteros en lugar de datos para optimizar la compartición.

parámetro	significado
<code>CLONE_FS</code>	compartir sistema de ficheros
<code>CLONE_VM</code>	compartir memoria virtual
<code>CLONE_SIGHAND</code>	compartir manejadores de señales
<code>CLONE_FILES</code>	compartir los ficheros abiertos

Bibliotecas de hebras

- Pthreads (POSIX threads/estándar IEEE 1003.1c).
 - API para la creación y sincronización de hebras.
 - Especificación \neq implementación.
 - Disponible en la mayoría de los SOs.
- Hebras Win32.
- Hebras Java.

Pthreads: API

```
#include <pthread.h>
```

función	descripción
<code>pthread_create(id, attr, func, arg)</code>	crea una nueva mediante la ejecución de la función <code>func</code>
<code>pthread_exit(val)</code>	finaliza un hebra y devuelve un valor para otra que quiera esperarla
<code>pthread_join(id, val)</code>	espera una hebra que finaliza y recupera el valor que devuelve
<code>pthread_self()</code>	devuelve el identificador de la hebra con tipo <code>pthread_t</code>
<code>pthread_yield()</code>	ceder el procesador voluntariamente

Pthreads: ejemplo en C

```
#include <pthread.h>

void* hebra(void*)
{
    return NULL;
}

int main()
{
    pthread_t id;

    pthread_create(&id, NULL, hebra, NULL);
    pthread_join(id, NULL);
}
```

Pthreads: ejemplo en C++ (C++0x/C++11)

```
#include <iostream>
#include <thread>

void hello()
{
    std::cout << "hello world!" << std::endl;
}

int main()
{
    std::thread t(hello);
    t.join();
}
```