Hebras (Threads)

Agenda

1. Objetivos
2. Introducción
3. Programación multinúcleo
4. Modelos Multihebra
5. Bibliotecas de hebras

UNIVERSIDAD DEL BÍO-BÍO

Objetivos

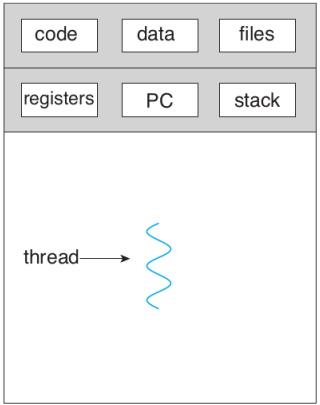
- Presentar el concepto de hebra, una unidad fundamental de utilización de la CPU que conforma los fundamentos de los sistemas informáticos multihebra.
- Describir los principales beneficios y los desafíos del diseño de procesos multihebras.
- Diseñar programas multihebras usando alguna de las APIs de hebras disponibles.

Introducción

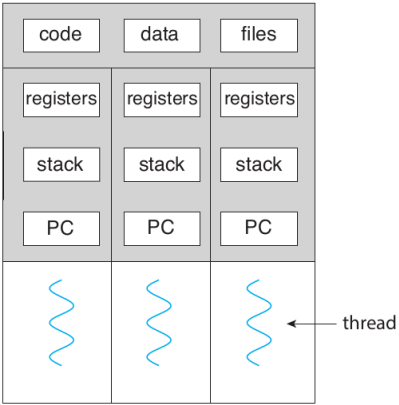
- Unidad básica de utilización de la CPU.
- Comprende:
  - ID de hebra
  - Contador de programa
  - Conjunto de registros y pila
- Comparte con otras hebras del mismo proceso:
  - Sección de código
  - Sección de datos
  - Otros recursos del SO (archivos abiertos).
- Proceso tradicional tiene una sola hebra de control. **Monohebra (single-threaded)**.
- Proceso con múltiples hebras de control. **Múltihebra (multithreaded)**.

UNIVERSIDAD DEL BÍO-BÍO

Hebras



single-threaded process



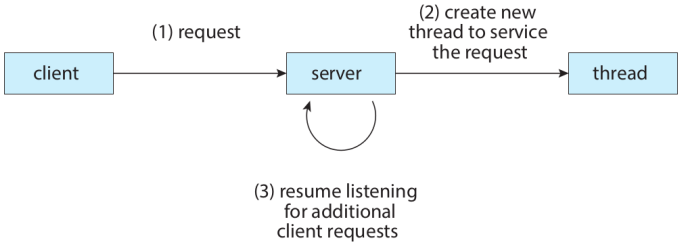
multithreaded process

Motivación

- Muchas aplicaciones hoy en día son multihebras. Es decir, son procesos que tienen varias hebras de control. Por ejemplo:
  - Una aplicación que crea imágenes miniatura (thumbnails) desde una colección de imágenes puede usar hebras separadas por cada imagen.
  - Un navegador web podría tener una hebra para mostrar imágenes o texto mientras otra hebra recibe datos desde la red.
  - Un procesador de texto puede tener una hebra para mostrar gráficos, otra hebra para capturar lo que digita un usuario y otra para verificar ortografía.

Motivación

- Se puede requerir una sola aplicación para realizar varias tareas similares.
  - Un servidor web acepta solicitudes de clientes por páginas web, imágenes, sonido, etc.
  - Sometido a gran carga puede tener miles de solicitudes concurrentes.
  - Si proceso tiene sóla una hebra, atiende una solicitud por vez.
  - Si proceso crea otros procesos, fork(), lleva tiempo y hace uso intensivo de recursos.
  - Si proceso es multihebra, por cada solicitud crea una nueva hebra.



Motivación

- Muchos sistemas operativos son también multihebras.
  - Linux cuando se inicia crea varias hebras (kernel threads) para tareas específicas: administración de dispositivos, administración de memoria, manejo de interrupciones, entre otras.
- Las aplicaciones también se pueden diseñar para aprovechar las capacidades de procesamiento en los sistemas multicore. Realizar tareas intensivas en CPU en paralelo.

UNIVERSIDAD DEL BÍO-BÍO

Ventajas

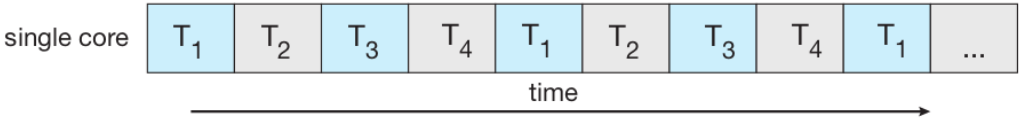
- **Capacidad de respuesta**: En una aplicación interactiva permite que programa continúe ejecutandose aunque parte de él esté bloqueado o E/S muy larga.
- **Compartición de recursos**: Hebras comparten memoria y recursos del proceso al que pertenecen. Aplicación tenga varias hebras diferentes en el mismo espacio de direcciones.
- **Economía**: Asignación de memoria y recursos en creación de procesos es costosa. Dado que hebras comparten recursos del proceso, es mas económico crear y realizar cambios de contexto entre hebras.
- **Escalabilidad**: Los beneficios de las multihebras pueden ser aún mayores en una arquitectura multiprocesador, donde las hebras pueden estar funcionando en paralelo en diferentes núcleos de procesamiento. Un proceso single-threaded puede ejecutarse en un solo procesador, independientemente de cuántos estén disponibles.

UNIVERSIDAD DEL BÍO-BÍO

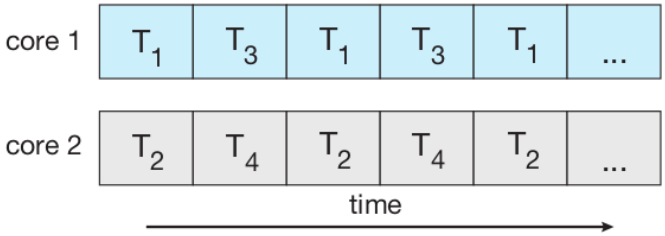
Programación multinúcleo

- Los sistemas single-CPU han evolucionado a sistemas multi-CPU (y luego a sistemas multi-core) en respuesta a la necesidad de un mayor rendimiento computacional.
- La programación multihebra proporciona un mecanismo para un uso más eficiente de estos múltiples núcleos y una concurrencia mejorada.
- Considere una aplicación con cuatro hebras:
  - En un sistema con un solo núcleo de computación, la concurrencia significa que la ejecución de las hebras se entrelazará en el tiempo.
  - En un sistema con múltiples núcleos, sin embargo, las hebras pueden ejecutarse en paralelo. El sistema puede asignar una hebra a cada núcleo.

Ejecución concurrente (sistema single-core)



Ejecución paralela (sistema multicore)

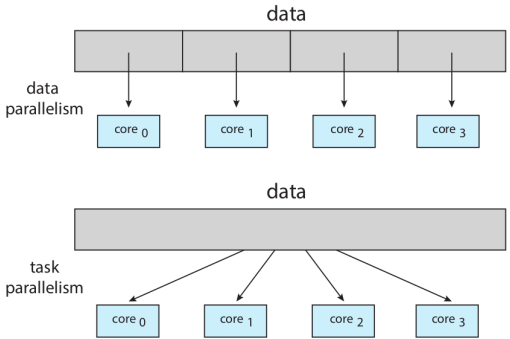


Desafíos programación multinúcleo

- En el diseño y/o modificación de programas existentes.
  - Identificación de tareas que puedan ejecutarse idealmente de manera independiente.
  - Lograr un balance en la carga de trabajo de cada tarea.
  - Dividir los datos para ser manipulados en diferentes núcleos.
  - Identificar dependencia de datos que pueden ser accedidos por dos o más tareas. Si una tarea depende de los datos de otra tarea, se debe asegurar un acceso sincronizado.
  - Las pruebas y depuración son más difíciles.

Tipos de paralelismo

- Paralelismo de datos. Distribuir subconjuntos de los datos en múltiples núcleos y realizar la misma operación en cada núcleo.
- Paralelismo de tareas. Implica distribuir tareas (hebras) en múltiples núcleos.
- Se pueden usar ambas estratégias, no son excluyentes.



Modelos Multihebra

- En la práctica el soporte de hebras puede proporcionarse como:
  - **Hebras de usuario**: soporte por encima del kernel, gestión de hebras sin soporte del kernel. (POSIX Pthreads, Win32 threads, Java threads).
  - **Hebras del kernel**: kernel soporta y gestiona directamente las hebras. (Windows, Linux, macOS).
- Debe existir una relación entre las hebras de usuario y las del kernel:
  - Modelo muchos-a-uno.
  - Modelo uno-a-uno.
  - Modelo muchos-a-muchos.

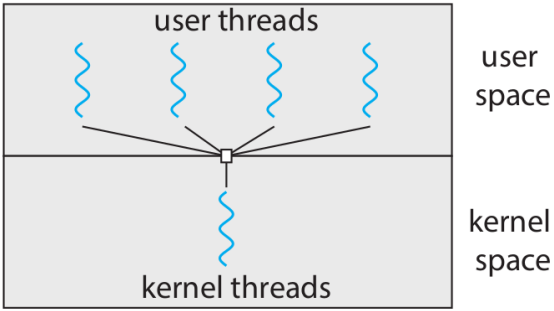
UNIVERSIDAD DEL BÍO-BÍO

Modelo Muchos-a-uno

- Asigna múltiples hebras del nivel de usuario a una hebra del kernel.
- Gestión de hebras en espacio de usuario, mediante biblioteca de hebras.
- Proceso completo se bloquea si una hebra realiza una llamada bloqueante al sistema.
- Sólo una hebra puede acceder al kernel a la vez, no podrán ejecutarse varias hebras en paralelo.

UNIVERSIDAD DEL BÍO-BÍO

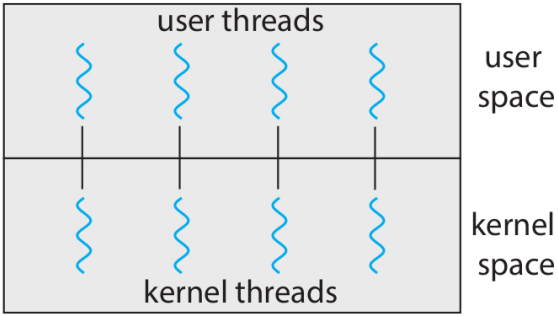
Modelo Muchos-a-uno



Modelo Uno-a-uno

- Asigna cada hebra de usuario a una hebra del kernel.
- Mayor concurrencia que muchos-a-uno, permitiendo que otra hebra se ejecute mientras otra hace un llamado bloqueante.
- Permite ejecución paralela en varios procesadores.
- Para no afectar rendimiento, se restringue el número de hebras soportadas por el sistema.
- Linux, Windows.

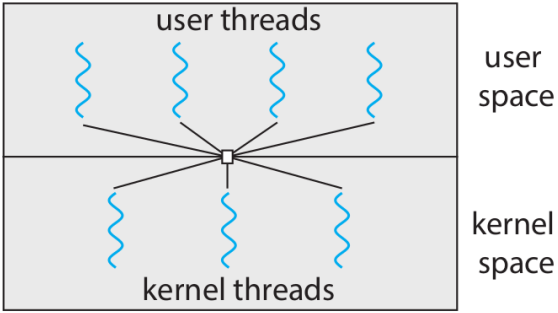
Modelo Uno-a-uno



Modelo Muchos-a-muchos

- Multiplexa muchas hebras de usuario sobre un número menor o igual de hebras de kernel.
- Se pueden crear tantas hebras de usuario como sean necesarias y las correspondientes hebras del kernel pueden ejecutarse en paralelo en un multiprocesador.

Modelo Muchos-a-muchos



Bibliotecas de hebras

- Proporcionan al programador una API para crear y gestionar hebras.
- Dos formas de implementar estas bibliotecas:
  - Enteramente en el espacio de usuario, sin ningún soporte del kernel. Estructuras de datos y código de biblioteca en el espacio de usuario. Funciones producen llamadas locales y no a sistema.
  - Soportada directamente por el SO, código y estructuras de datos de la biblioteca se encuentran en el espacio del kernel. Funciones producen llamadas al sistema.
- Las tres principales bibliotecas:
  - POSIX Pthreads (nivel usuario/kernel).
  - Librería de hebras de Windows (nivel kernel).
  - Java thread API (usa librería de hebras disponibles en el sistema operativo).

UNIVERSIDAD DEL BÍO-BÍO

Pthreads[1]

- Basado en el estándar POSIX (IEEE 1003.1c) que define una API para la creación y sincronización de hebras.
- Standard UNIX threading API.
- Más de 60 funciones: pthread_create, pthread_join, pthread_exit, etc.
- Relativamente de bajo nivel.
- Manejo explícito de las hebras (creación, eliminación de hebras, sincronización).
- Linux, macOS, implementan la especificación Pthreads.

[1] POSIX Threads Programming: `https://computing.llnl.gov/tutorials/pthreads/`

UNIVERSIDAD DEL BÍO-BÍO

OpenMP

- Extensión de lenguaje basado en (directivas de compilación, biblioteca de funciones.)
- Disponible para C/C++ y Fortran.
- Énfasis en high-performance computing (HPC).
- Alto nivel de abstracción.
- OpenMP: `https://www.openmp.org/`

```
#pragma omp parallel for
for( i = 0; i < n; i++ )
  z[i] = a * x[i] + y[i]
```

UNIVERSIDAD DEL BÍO-BÍO