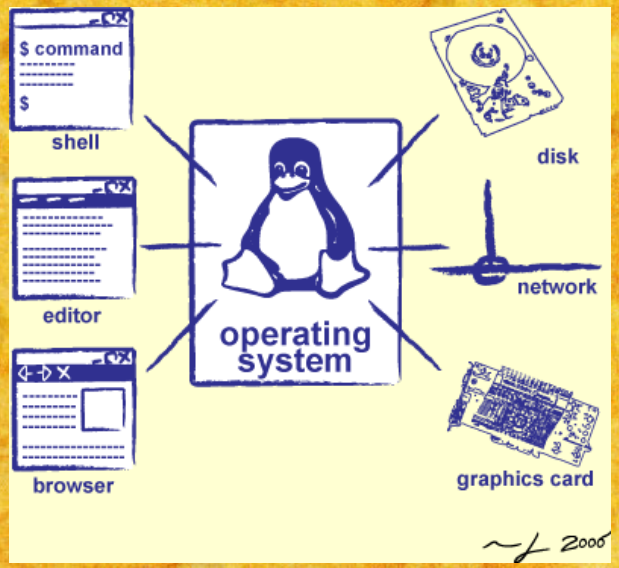
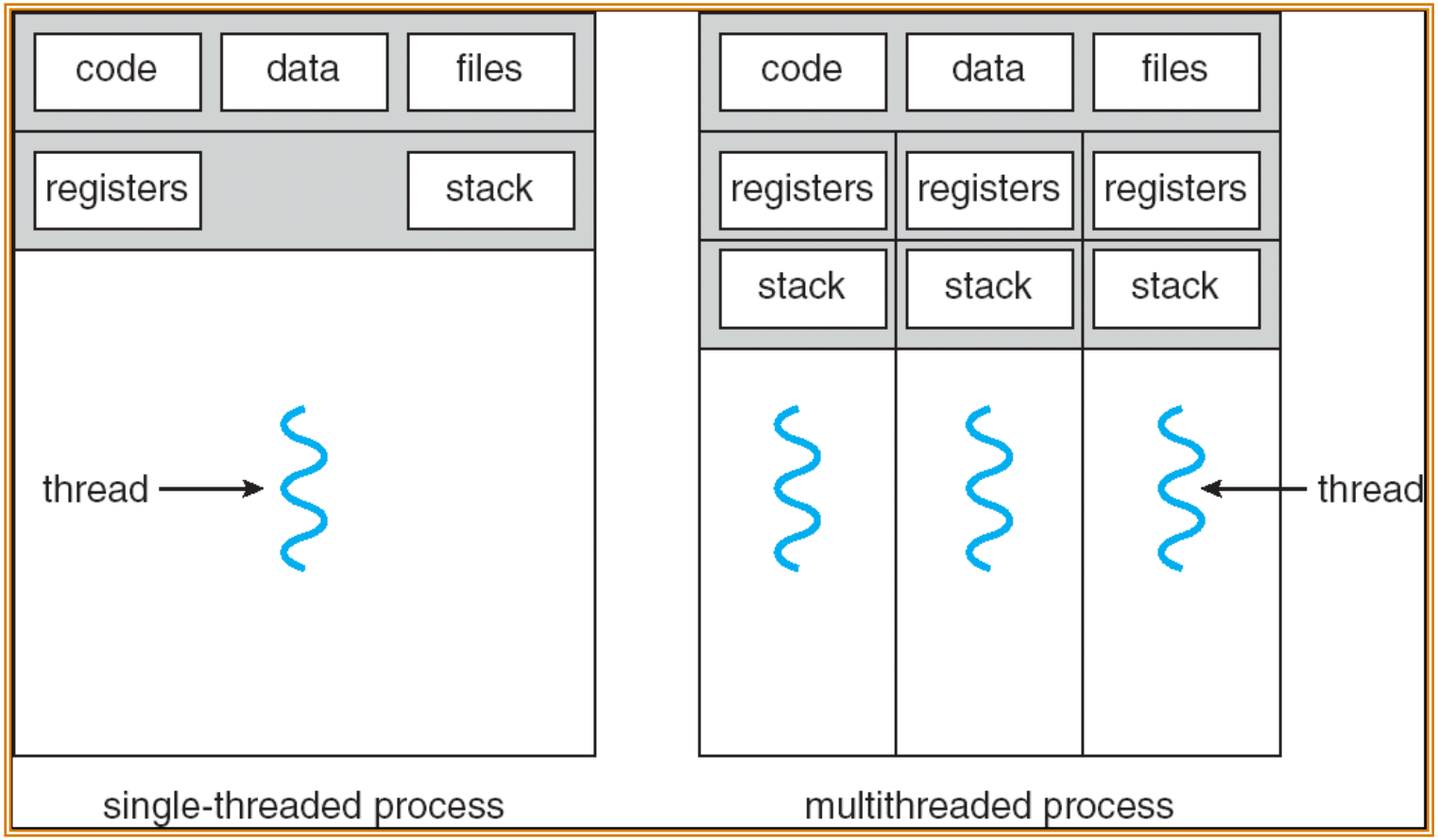
4.- Threads


$ command
$
shell
editor
browser
operating system
disk
network
graphics card
2006

Capítulo 4: Threads

- Modelos Multithreads
- Threads
- Pthreads
- Threads de Windows XP
- Threads de Linux
- Threads de Java

Procesos Uni y Multi-hilos



Beneficios

- Rapidez de respuesta
- Compartir recursos
- Economía
- Utilización de Arquitecturas MP

Threads de Usuario

- Manejo de threads hecho por la biblioteca de threads a nivel usuario
- Bibliotecas principales:
  - Pthreads POSIX
  - Threads Win32
  - Java threads

Threads del Kernel

- Manejadas por el Kernel
- Ejemplos
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

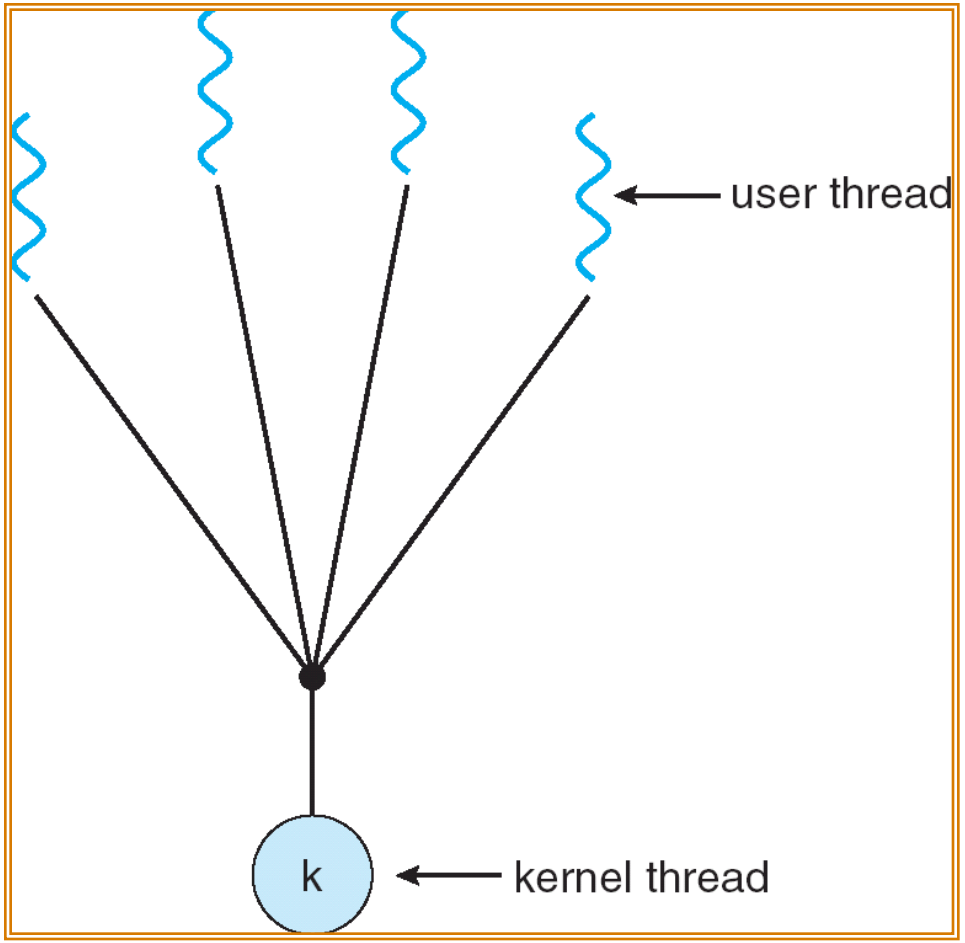
Modelos Multi-hilos

- Muchos-a-uno
- Uno-a-uno
- Muchos-a-muchos

Muchos-a-uno

- Muchos threads nivel usuario se mapean a un solo thread del kernel
- Ejemplos:
  - Solaris Green Threads
  - GNU Portable Threads

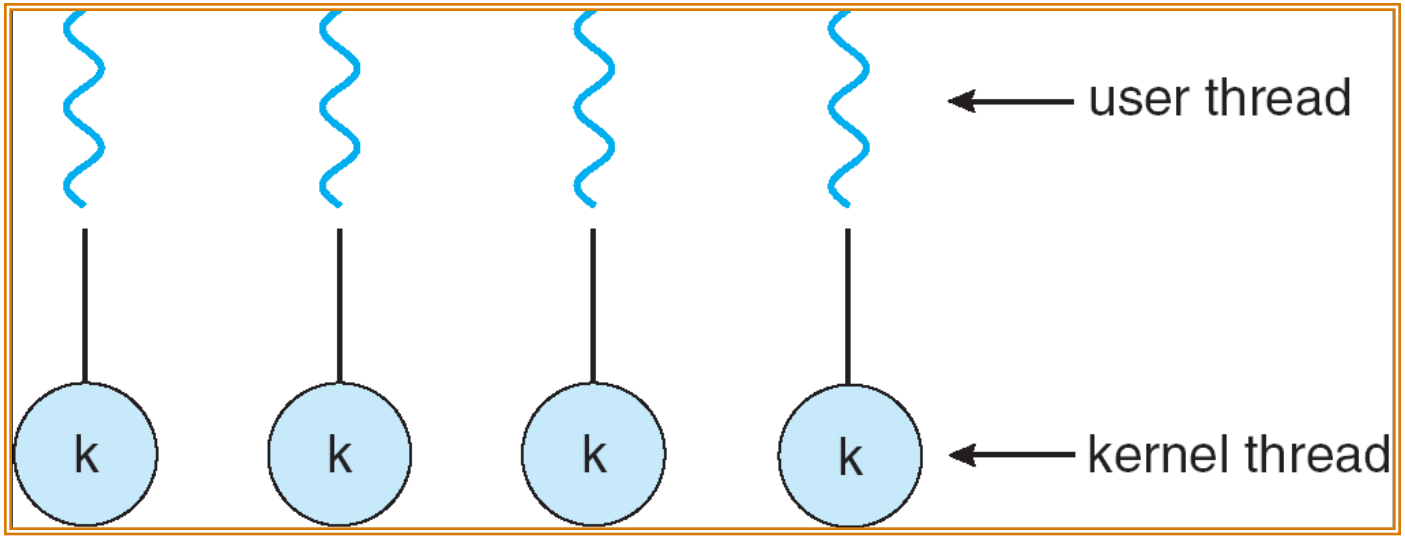
Modelo Muchos-a-uno



Uno-a-uno

- Cada thread nivel usuario se mapea a un thread del kernel
- Ejemplos
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 y posterior

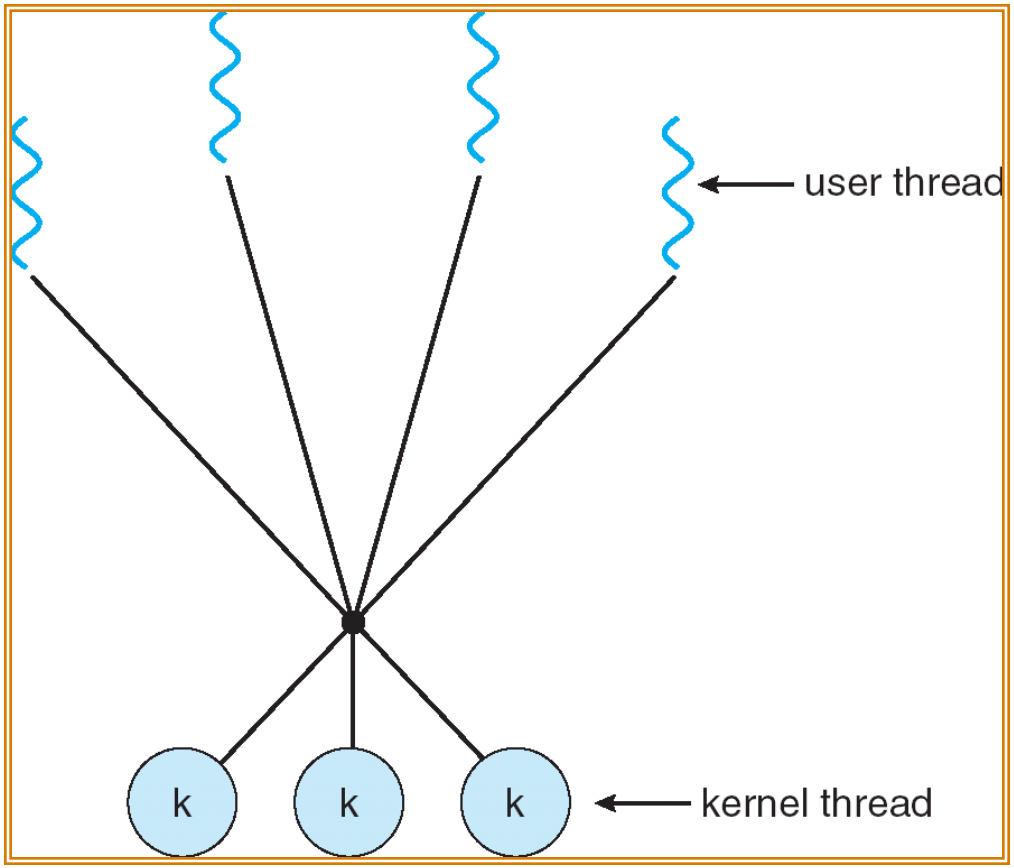
Modelo Uno-a-uno



Modelo Muchos-a-muchos

- Muchos threads de usuario se mapean a muchos threads de kernel
- Le permite al SO que cree un número suficiente de threads de kernel
- Solaris anterior a versión 9
- Windows NT/2000 con el paquete *ThreadFiber*

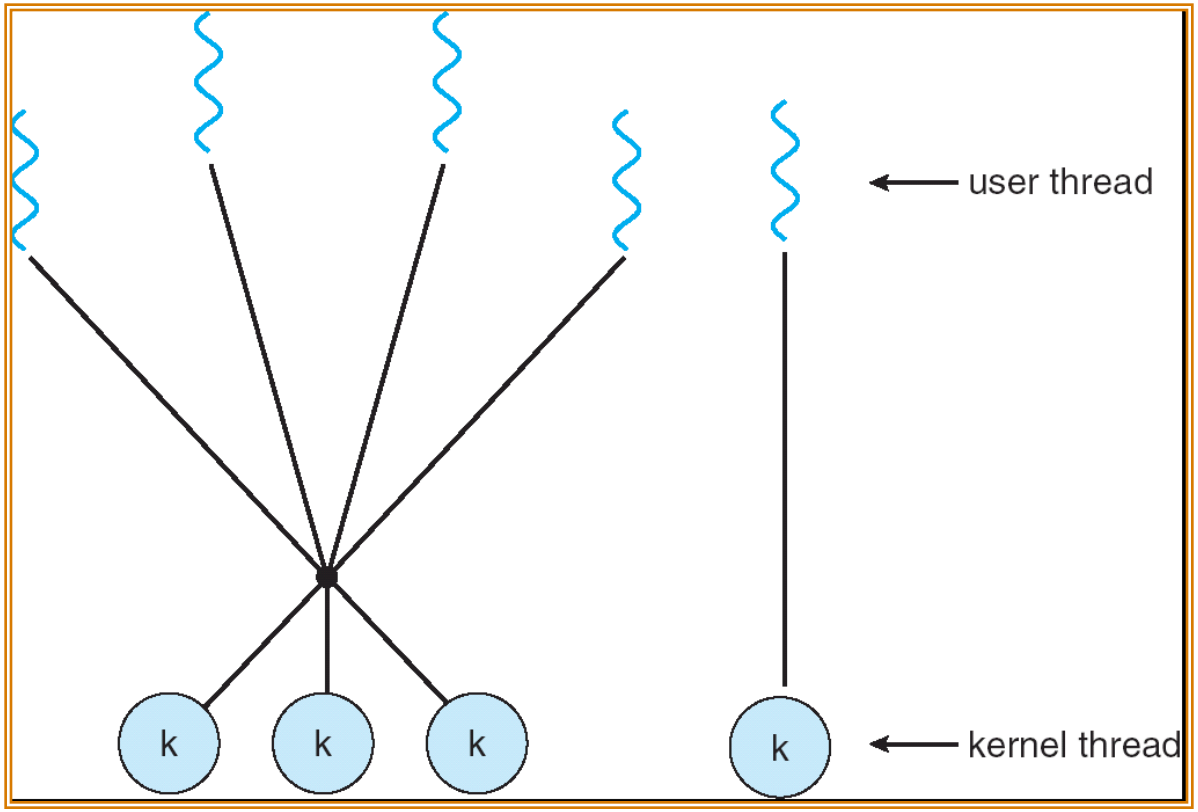
Modelo Muchos-a-muchos



Modelo de Dos Niveles

- Similar a M:M, excepto que permite que un thread de usuario se vincule a un thread del kernel
- Ejemplos
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 y anteriores

Modelo de Dos Niveles



Problemas con Threads

- Semántica de **fork()** y **exec()**
- Cancelación de threads
- Manejo de señales
- Conjuntos de threads
- Datos específicos de un thread
- Activación del despachador

Semántica de fork() y exec()

- ¿ Duplica **fork()** solo el thread que lo invoca o todos los threads?

Cancelación de Threads

- Terminar un thread antes de que acabe
- Dos ideas generales:
  - **Cancelación Asíncrona** – termina el thread inmediatamente
  - **Cancelación Diferida** – permite al thread verificar periódicamente si debe ser cancelado

Manejo de Señales

- UNIX usa señales para notificar a un proceso que han ocurrido eventos
- Un manejador de señales (**signal handler)** se usa para procesar señales
  1. Señal generada por un evento particular
  2. Señal entregada a un proceso
  3. Señal procesada (handled)
- Opciones:
  - Entregar la señal al thread que corresponde
  - Entregar la señal a todos los threads en el proceso
  - Entregar la señal a ciertos threads en el proceso
  - Asignar un thread específico para recibir todas las señales para el proceso

Thread Pools

- Crear threads en un “pool”, donde esperan a que se les asigne trabajo
- Ventajas:
  - Ligeramente más rápido que crear un thread nuevo
  - El número de threads en la(s) aplicación(es) está limitado al tamaño del pool

Datos del Thread

- Cada thread tiene su propia copia de los datos
- Útil cuando no se tiene control en el proceso de creación de threads (i.e., thread pool)

Activaciones del Scheduler

- Ambos modelos M:M y Dos-Niveles requieren comunicación para mantener un número apropiado de kernel threads asignadas a la aplicación
- Proporcionan **upcalls** – un mecanismo de comunicación del kernel a la biblioteca de threads
- Esta comunicación le permite a una aplicación mantener un número correcto de threads

Pthreads

- API Estándar (IEEE 1003.1c) para creación y sincronización de threads
- El API especifica el comportamiento de la biblioteca de threads (la implementacón se deja a los desarrolladores)
- Común en SOs UNIX (Solaris, Linux, Mac OS X)

Windows XP Threads

- Implementa mappeo one-to-one
- Cada thread contiene
  - Thread id
  - Conjunto de Registros
  - User and kernel stacks separados
  - Área de datos privada
- El conjunto de registros, stack y datos privados son el **contexto** de un thread
- Las estructuras de datos primarias de un thread incluyen:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Threads de Linux

- En Linus se llaman *tasks* en lugar de *threads*
- Se crean con el system call **clone()**
- **clone()** le permite a un child task compartir el espacio de direcciones del parent task (process)

Java Threads

- Manejadas por la JVM
- Pueden ser creadas por:
  - Extención de la clase Thread
  - Implementación de la interfaz Runnable