# MultiThreading

César Pedraza Bonilla

Universidad Nacional

*capedrazab@unal.edu.co*

13 de abril de 2015

# Overview

# Multithreading

- Hace referencia a la capacidad de un S.O. de soportar múltiples hilos de ejecución concurrentes en un mismo proceso.
- Un proceso se comporta también como una unidad de asignación de recursos y de protección a los hilos.
- En un sistema multi-hilo cada hilo tiene: un estado, contexto guardado, pila de ejecución, almacenamiento local, memoria compartida con otros hilos.

# Multithreading

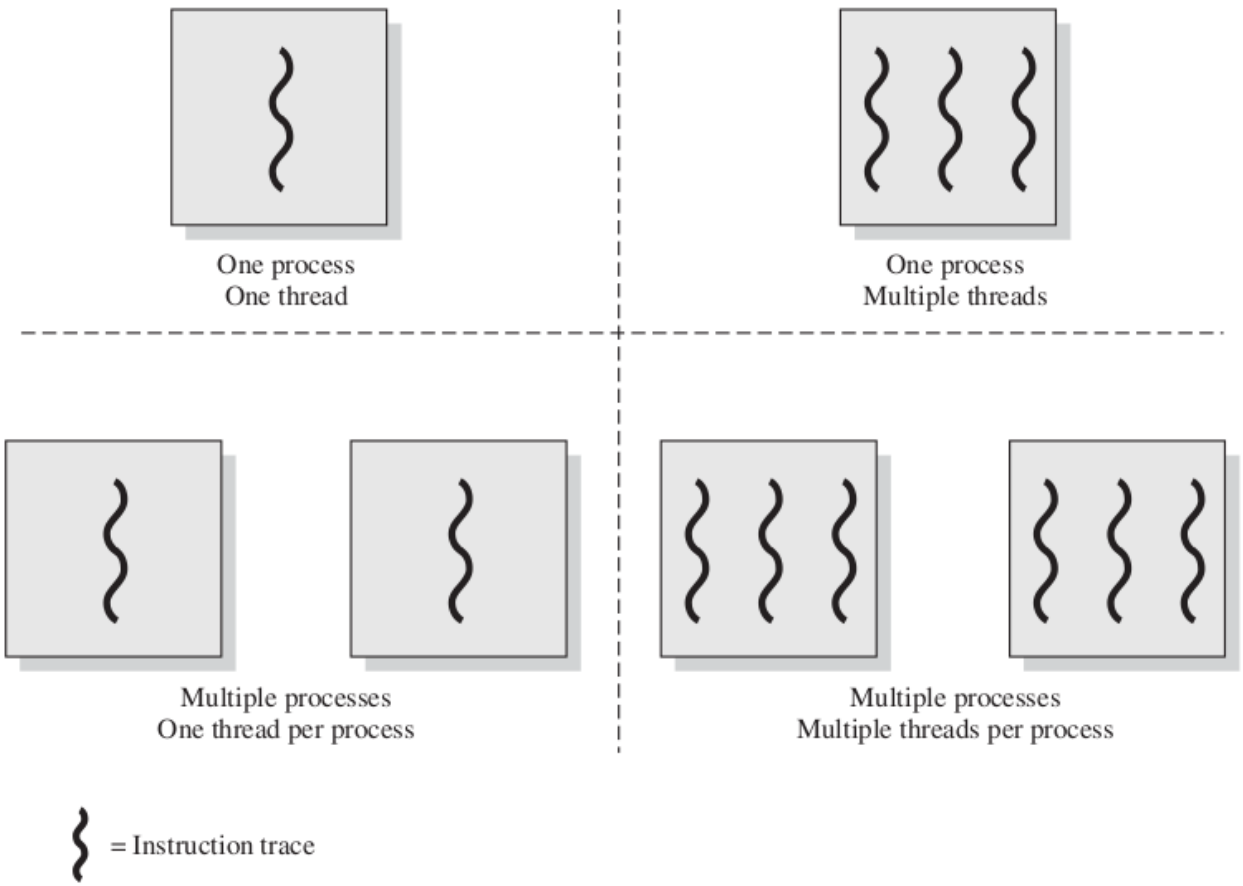


Figura: Hilos y procesos.[1]

[1] Tomado de Stallings, Operating Systems.

# Multithreading

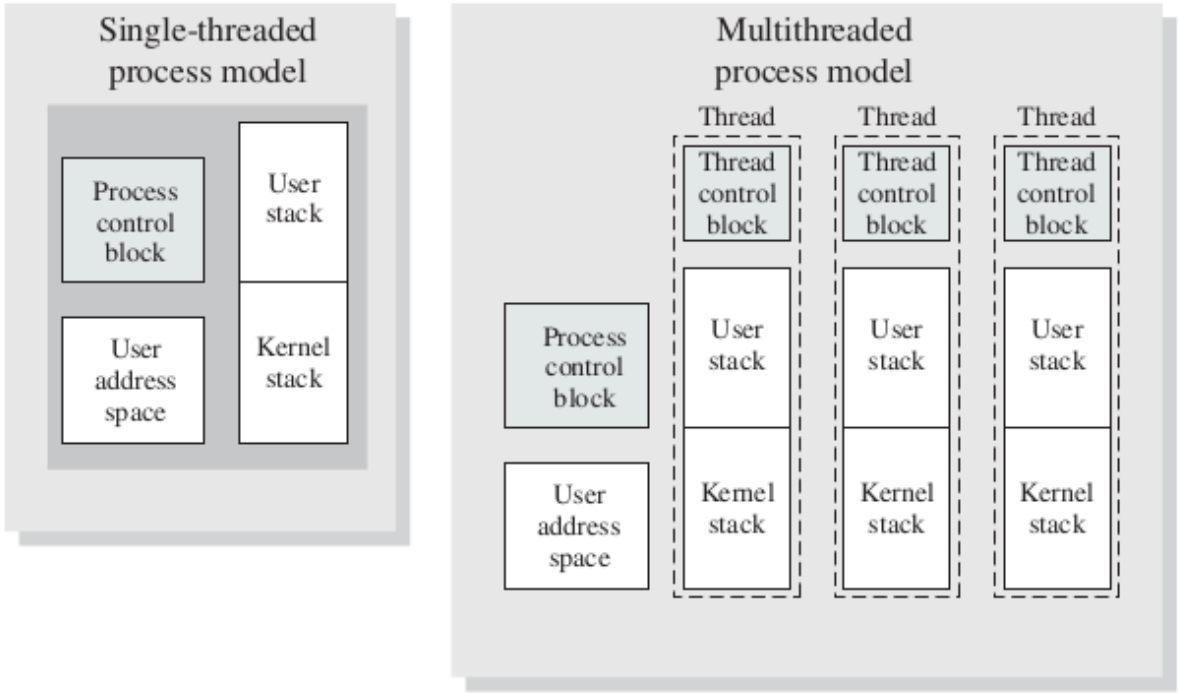


Figura: Hilos y procesos.[2]

[2] Tomado de Stallings, Operating Systems.

# Multithreading

- El espacio de usuario de un proceso es compartido a todos los hilos que se creen a partir de dicho proceso.
- Los *kernel y user stack* controlan los llamados a rutina e interrupciones dentro de cada hilo.

# Multithreading

Beneficios del uso de hilos.

- La creación de un hilo es más rápida que la creación de todo un proceso. Estudios revelan que hay casos en los que es hasta 10 veces más rápido.
- Es más rápido terminar un hilo que un proceso.
- El cambio de contexto es más rápido con hilos que con procesos.
- La comunicación entre hilos es más rápida que entre procesos (memoria compartida sin uso de kernel).

# Multithreading

Ejemplos de uso de hilos:

- Trabajo de primer y segundo plano. Por ejemplo si se trabaja con un programa con parte gráfica, un hilo controla los menús y otro las funciones específicas.
- Procesamiento asíncrono. Por ejemplo cuando se requiere realizar una tarea cada cierto tiempo, como guardar datos de RAM a disco.
- Velocidad de procesamiento. Cuando se requiere aumentar el rendimiento en tiempo de respuesta.
- Modularidad de un programa. Para separar tareas que claramente deben estar separadas.

# Multithreading

Estados de un hilo.

- Creado *spawn*. Cuandos se crea un proceso, es posible que sean creados sus hilos también, si así se ha solicitado.
- Bloqueado. Cuando se requiere que un hilo espere a un evento.
- Desbloqueado. Cuando al estar a bloqueado pasa a la cola y queda listo.
- Finalizado. Cuando un hilo termina su ejecución y su registro de contexto y las pilas *kernel and user stack* son eliminadas.

No se especifica si está en *swap*, dado que si un proceso es suspendido a swap, todos sus hilos son pasados a dicho estado.

# Tipos de hilos

Existen dos tipos de hilos:

- User Level Threads ULT. Todo el manejo de los hilos es realizada por la aplicación, sin el uso del kernel. *pthread* Todo el proceso de creación, gestión y finalización de los hilos es llevada a cabo en espacio de usuario.
- Kernel Level Threads KLT. *kthread_run*

# Tipos de hilos

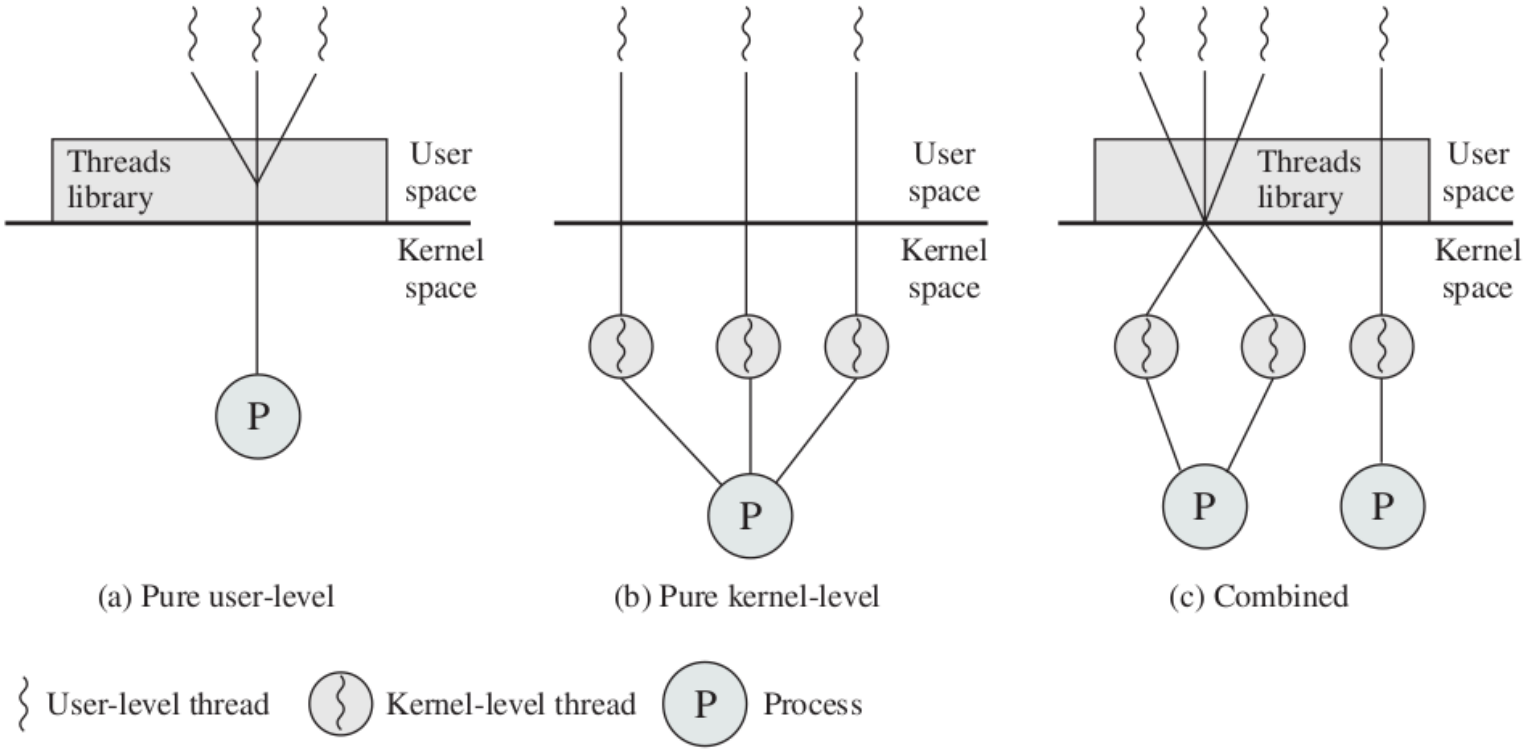


Figura: Hilos a nivel de usuario y a nivel de kernel.[3]

[3] Tomado de Stallings, Operating Systems.

# Tipos de hilos

Ventajas de los ULT:

- El cambio de contexto de los hilos no requiere de la intervención del kernel, por tanto, no hay cambios de modos en el S.O.
- Una aplicación podría tener su propio algoritmo de planificación *scheduling*.
- Los ULT pueden correr sobre cualquier S.O.

Desventajas:

- Debido a que la mayoría de las llamadas a sistema son bloqueantes, si un hilo es bloqueado, los demás también lo estarán.
- En un entorno *multicore* el S.O. podría asignar sólo un núcleo a un proceso, y por consiguiente sus hilos sólo se ejecutarán en dicho núcleo.

# Tipos de hilos

Ventajas de los KLT.

- Todo el trabajo de creación y gestión de los hilos es realizada por el kernel.
- El kernel puede mantener diferentes hilos en diferentes núcleos.
- Las propias rutinas del kernel pueden ser lanzadas en diferentes hilos.

Desventajas de los KLT:

- El control de los hilos pasa al kernel, lo que implica que haya un cambio en el modo de ejecución.