Concurrencia y paralelismo a nivel del Kernel

La **concurrencia** y el **paralelismo** son conceptos fundamentales en sistemas operativos y programación de alto rendimiento. La concurrencia se refiere a la capacidad de ejecutar múltiples tareas de forma aparentemente simultánea, intercalando su ejecución. En cambio, el paralelismo se da cuando varias tareas se ejecutan realmente al mismo tiempo en múltiples núcleos de un procesador. A nivel de kernel, estos conceptos se aplican para gestionar procesos e hilos, coordinando la ejecución de múltiples tareas mediante planificación (*scheduling*) y sincronización.

El kernel es la parte central del sistema operativo que administra los recursos del sistema, incluidos los procesos e hilos. Un **proceso** es una instancia de un programa en ejecución con su propio espacio de memoria, mientras que un **hilo** es una unidad más ligera dentro de un proceso que comparte el mismo espacio de memoria con otros hilos del mismo proceso. El kernel crea y administra hilos mediante **context switching** (cambio de contexto), permitiendo que múltiples tareas avancen sin que se bloqueen mutuamente.

El kernel implementa hilos en un modelo **1:1** (un hilo de usuario por cada hilo de kernel) o en un modelo híbrido con multiplexación, donde varios hilos de usuario se mapean a menos hilos de kernel.

El kernel utiliza **algoritmos de planificación** para decidir qué proceso o hilo se ejecuta en un momento dado. Los algoritmos pueden ser **preemptivos** (donde el kernel interrumpe la ejecución de una tarea para ceder el control a otra) o **no-preemptivos**. Los sistemas modernos suelen usar la planificación por prioridades y **colas multilevel feedback** para gestionar tanto procesos en primer plano como tareas en segundo plano.

Por ejemplo, los sistemas basados en **Linux** utilizan el **Completely Fair Scheduler (CFS)**, que intenta distribuir el tiempo de CPU de manera justa entre todos los procesos, aprovechando los núcleos disponibles para ejecutar múltiples hilos en paralelo.

Cuando varios hilos necesitan acceder a recursos compartidos (como archivos, memoria o dispositivos), el kernel debe evitar condiciones de carrera y garantizar que los recursos se usen de forma consistente. Para esto, se utilizan mecanismos de **sincronización** como los **semaforos**, **mutexes** y **bloqueos por spin** (*spinlocks*).

- **Mutex**: Un mecanismo de bloqueo que garantiza la exclusión mutua; es decir, solo un hilo puede acceder a un recurso en un momento dado.
- **Spinlock**: Bloqueo activo donde el hilo espera en un bucle hasta que el recurso esté disponible, útil en entornos de baja latencia.
- **Semáforos**: Controlan el acceso basado en contadores, permitiendo que varios hilos accedan simultáneamente hasta un límite predefinido.

Estos mecanismos aseguran que los hilos no entren en secciones críticas simultáneamente, evitando problemas como *deadlocks* (bloqueos mutuos) y *starvation* (inaniación de un hilo).

El kernel permite el **paralelismo real** distribuyendo procesos e hilos en varios núcleos de la CPU. Cada núcleo puede ejecutar su propio hilo, lo que mejora el rendimiento en tareas como procesamiento de datos, gráficos y simulaciones científicas. Tecnologías como

Simultaneous Multithreading (SMT) (por ejemplo, **Hyper-Threading** en CPUs de Intel) permiten que cada núcleo maneje dos hilos simultáneamente, aprovechando mejor los ciclos ociosos del procesador.

Los kernels modernos también soportan la asignación de afinidad de procesos, donde un proceso o hilo se vincula a un núcleo específico para mejorar la **localidad de caché**. Además, en sistemas como Linux, se implementan **thread pools** y **kthreads** (hilos de kernel) para manejar tareas críticas del sistema sin bloquear otros procesos.

En Linux, los **hilos de kernel** (kthreads) se utilizan para gestionar tareas de bajo nivel, como la planificación de procesos, la gestión de memoria y la comunicación entre dispositivos. Estos hilos no están asociados directamente a procesos de usuario y se ejecutan en modo privilegiado.

Un ejemplo práctico de paralelismo a nivel de kernel es el manejo de **I/O asíncrono** (AIO). Cuando un proceso solicita datos de un disco, el kernel lanza un hilo que espera los datos y permite que el proceso continúe con otras tareas. Una vez que la operación I/O termina, el kernel notifica al proceso, evitando que este quede bloqueado esperando la respuesta.

En resumen, la concurrencia y el paralelismo a nivel de kernel son fundamentales para garantizar que los sistemas operativos modernos puedan manejar múltiples tareas de manera eficiente. Mediante la planificación cuidadosa de procesos, la sincronización de hilos y el uso efectivo de recursos multinúcleo, el kernel optimiza la ejecución de aplicaciones, mejorando tanto la capacidad de respuesta como el rendimiento global del sistema.