4. CONCURRENCIA Y TASKING

El objetivo de la concurrencia es facilitar programas complejos. Además, es más adecuado para los ordenadores multicore que se usan en la actualidad, puesto que se puede dedicar una hebra de un proceso a cada núcleo de procesador.

4.1. PROCESO Y HEBRA

La mayoría de sistemas operativos soporta multiprogramación.

➢ **Proceso:** Es un programa[1] en ejecución.

- Necesita recursos: tiempo de CPU, memoria, ficheros, dispositivos E/S.
- Los procesos están aislados: un proceso no puede acceder directamente a otro. El paso de un proceso a otro se realiza invocando al sistema operativo mediante llamadas al sistema.

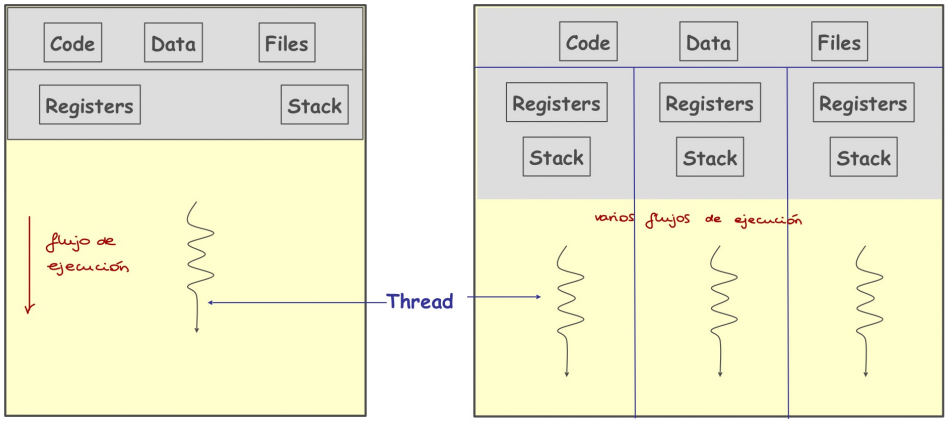
[1]Programa: Código y datos estáticos.

La imagen de un proceso en un sistema operativo muestra el código y el estado del proceso (memoria, registros…). El sistema operativo asigna "rodajas" de tiempo a cada proceso, y cuando ésta expira, pasa a ejecutar otro proceso. También puede haber cambio de proceso debido a la conexión de un dispositivo E/S o que una tarea con mayor prioridad esté lista para ejecutar.

El cambio de ejecutar un proceso a otro es lento, porque requiere cambiar las variables del estado (cambio de contexto) y además crear un proceso es lento.

➢ **Thread**: La mayoría de sistemas operativos soporta *multithreading*.

- Un proceso puede incluir un conjunto de *threads* (hebras) ejecutándose.
- Se ejecutan en concurrencia y comparten los recursos (no están aisladas). Son menos seguras que los procesos, porque si una falla, puede dañar los datos de otra. Pero es más rápido el cambio de una a otra (cambio de contexto), y es muy eficiente la comunicación entre hebras.
- En los sistemas embebidos es habitual que haya sólo hebras. En estos sistemas no suele ser necesario un gran nivel de protección o seguridad, porque todas las partes del software han sido desarrolladas por la misma empresa/entidad. La protección está pensada para la inclusión de programas desarrollados por otras entidades.



Single-threaded process Multithreaded process

4.2. TAREAS CONCURRENTES

Las tareas/threads son a menudo implementadas utilizando hebras:

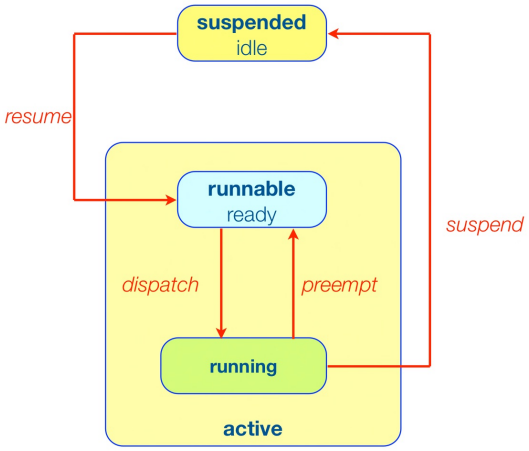
- Flujos de ejecución concurrentes que progresan de forma asíncrona.
- El tiempo de procesador se multiplexa entre las hebras activas
- Requiere el soporte de sistema operativo o run-time.

4.3. ESTADOS DE UNA TAREA

Una tarea puede estar en distintos estados.

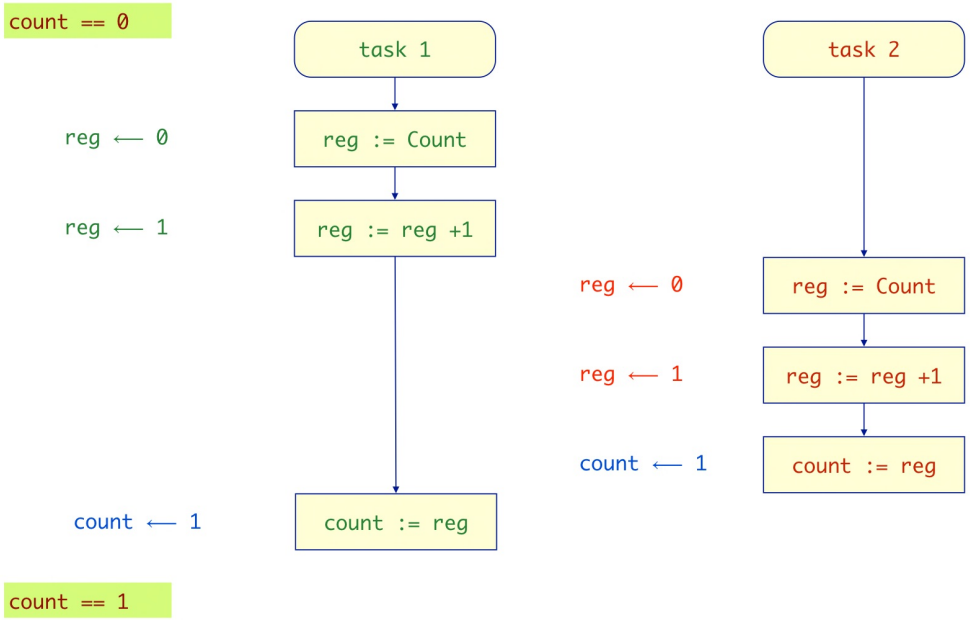
Las tareas ejecutables son enviadas para ejecución de acuerdo con un método de *scheduling*, el cual es implementado por el núcleo del sistema operativo o por el sistema de tiempo real.

- No ejecutando / suspendido.
- Listo para ejecutar (puede haber N tareas esperando).
- Ejecutando (el SO decide cuándo se va a ejecutar).



4.4. DATOS COMPARTIDOS

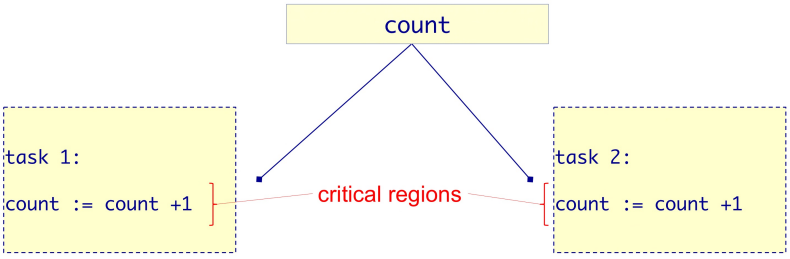
Es casi imposible hacer programas complejos con tareas independientes, sin comunicarse. Según cuándo se produzcan los cambios de contexto, podría dar lugar a error. Ejemplo: contador compartido.



El resultado podía ser 1 o 2 dependiendo de las velocidades de ejecución de cada tarea. Dicho programa es incorrecto, pues cada ejecución es diferente. Para solucionar el problema, se impone la exclusión mutua.

Región crítica: Segmento de código en el que se accede a un dato compartido.

Exclusión mutua: Si una tarea está en una región crítica de una variable, ninguna hebra puede acceder a dicha variable.



4.5. OBJETOS PROTEGIDOS EN ADA

Los objetos protegidos encapsulan los datos y operaciones compartidos. La exclusión mutua se genera automáticamente.

Son instancias de (posiblemente anónimos) tipos protegidos.

Un objeto protegido, o monitor, impone que si hay una tarea ejecutando una operación del monitor, ninguna tarea puede ejecutar otra operación.