Proyecto FPMADdigital

Recursos digitales y multimedia para Formación Profesional

Comunidad de Madrid

Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial

CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"

Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

FPMADdigital
Recursos digitales y multimedia para Formación Profesional

CFGS Desarrollo de Aplicaciones Multiplataforma

módulo profesional

0490 - Programación de Servicios y Procesos

unidad didáctica

02 Programación concurrente y asíncrona

resultados de aprendizaje

02 Desarrolla aplicaciones compuestas por varios hilos de ejecución

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

Resultados de aprendizaje y unidades didácticas

Resultados de aprendizaje y unidades didácticas

RESULTADOS DE APRENDIZAJE					UNIDAD DIDÁCTICA
1	2	3	4	5	
X					1.- Programación multiproceso
	X				2.- Programación concurrente y asíncrona
		X			3.- Programación de comunicaciones en red
			X		4.- Generación de servicios en red
				X	5.- Técnicas de programación segura

Unidades didácticas y materiales asociados

Unidades didácticas y materiales multimedia

RRAA					UU.DD	Material Multimedia
1	2	3	4	5		
X					1.- Programación multiproceso	1.1 Contenidos básicos 1.2 Ejemplos aplicados
	X				2.- Programación concurrente y asíncrona	2.1 Contenidos básicos 2.2 Ejemplos aplicados
		X			3.- Programación de comunicaciones en red	3.1 Contenidos básicos 3.2 Ejemplos aplicados
			X		4.- Generación de servicios en red	4.1 Contenidos básicos 4.2 Ejemplos aplicados
				X	5.- Técnicas de programación segura	5.1 Contenidos básicos 5.2 Ejemplos aplicados

Repositorios de materiales y prácticas

Repositorio de materiales y prácticas

Todos los proyectos mostrados, así como otros materiales utilizados en las unidades didácticas los podrás encontrar completos en:

https://github.com/joseluisgs/FP-NextGen-ProgramacionServiciosProcesos

Cualquier error o propuestas de mejora se publicarán en el repositorio indicado.
Gracias por tu colaboración.

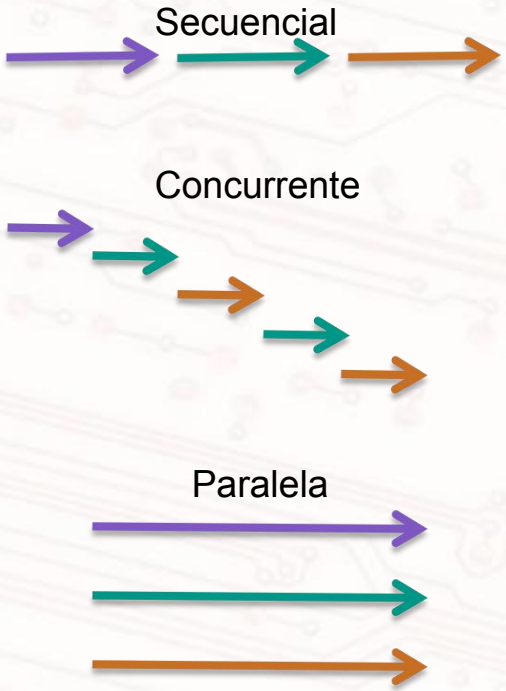
Contenidos

1. Concurrencia y Asincronía
2. Recursos compartidos
3. Programación multihilo
4. Sincronización y comunicación
5. Ejemplos

Concurrencia y Asincronía

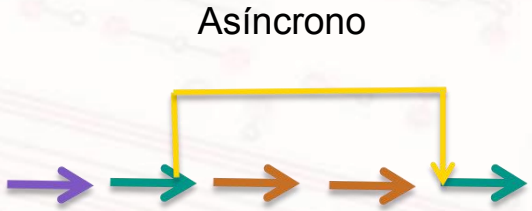
Concurrencia y Asincronía

- **Prog. sencuencial**: una instrucción sigue a otra instrucción después de que esta haya finalizado. Un solo núcleo y un hilo por núcleo.
- **Prog. concurrente**: múltiples instrucciones de un programa funcionen al mismo tiempo. Múltiples núcleos, múltiples hilos por núcleo.
- **Prog. paralela**: utilización de técnicas de programación y arquitecturas hardware para que distintas instrucciones se ejecuten a la vez y al mismo tiempo. Múltiples núcleos.



Sincronía y Asincronía

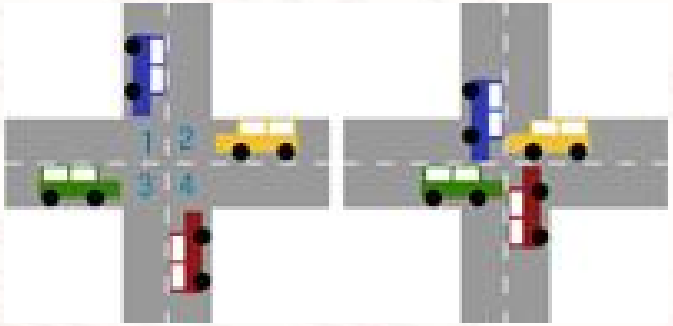
- **Sincronía**: toda la operación se ejecuta de forma secuencial y, por tanto, debemos esperar a que se complete para procesar el resultado, por lo que puede ser bloqueante.
- **Asincronía**: la finalización de la operación se señaliza más tarde, mediante un mecanismo específico como por ejemplo un callback, una promesa o un evento lo que hace posible que la respuesta sea procesada en diferido. Como se puede adivinar, su comportamiento es no bloqueante.



Recursos compartidos

Recursos compartidos

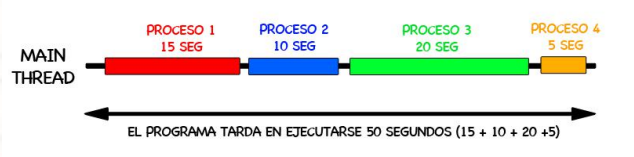
- Un proceso/hilo entra en condición de **competencia** con otro, cuando ambos necesitan el mismo recurso, ya sea de forma exclusiva o no; por lo que será necesario utilizar mecanismos de sincronización y comunicación entre ellos.
- Se llama **sección crítica** al conjunto de instrucciones en las que el proceso utiliza un recurso y que se deben ejecutar de forma exclusiva con respecto a otros procesos competidores por ese mismo recurso, asegurando la **exclusión mutua**.
- Se dice que un proceso hace un lock (**bloqueo**) sobre un recurso cuando ha obtenido su uso en exclusión mutua.
- Deadlock o **interbloqueo**, se produce cuando los procesos no pueden obtener, nunca, los recursos necesarios para continuar su tarea. El interbloqueo es una situación muy peligrosa, ya que puede llevar al sistema a su caída o cuelgue.

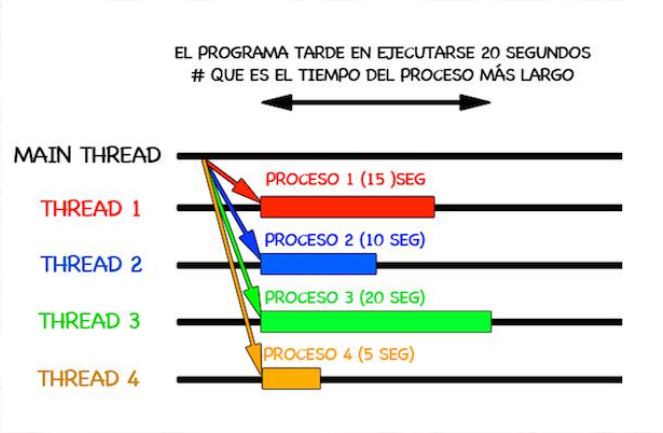


Programación multihilo

Programación multihilo

- Un **hilo** es un objeto con capacidad de correr en forma concurrente y compartir el espacio de direcciones con otros hilos dentro del mismo proceso. Por lo tanto un proceso puede tener varios hilos ejecutando acciones concurrentes. Al menos tiene uno (main)
- Está limitado por el número de hilos/núcleos soportados por el sistema.
- Un hilo puede compartir con otros hilos del mismo proceso los siguientes recursos:
  - Código.
  - Datos (como variables globales).
  - Otros recursos del sistema operativo, como los ficheros abiertos y las señales.
- Seguro que te estarás preguntando "si los hilos de un proceso comparten el mismo espacio de memoria, ¿qué pasa si uno de ellos la corrompe?" La respuesta es, que los otros hilos también sufrirán las consecuencias.

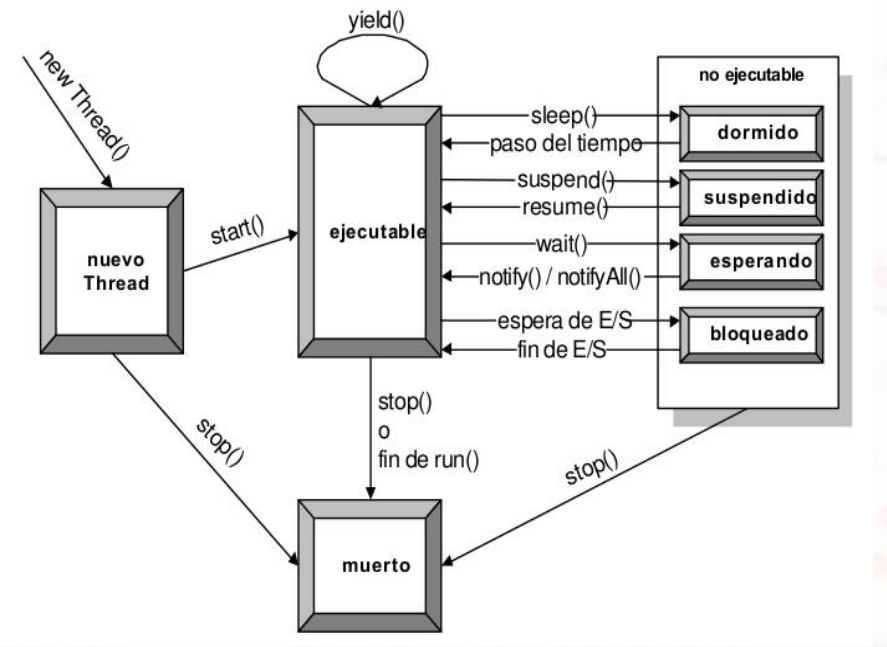




StackOverflow

Estado de un hilo

- Creación (**LISTO**) de ahí puede pasar EN **EJECUCION** o no, esto dependerá del Dispacher.
- Una vez EN **EJECUCION** procesará su código hasta que se le acabe el tiempo asignado.
- Puede pasar a **LISTO**, **ESPERA**, **DORMIDO**, **SUSPENDIDO** y **BLOQUEADO**.
- Pasará a **DORMIDO** cuando se invoque el método sleep() y de ahí a **LISTO**.
- Pasará a **BLOQUEADO** cuando tenga que sufrir una espera debida a una instrucción bloqueante, saldrá de este estado en cuanto termine.
- El estado **SUSPENDIDO** es para aquellos hilos que han quedado suspendidos y deben resumirse.
- Pasará a **EN ESPERA** cuando alguien invoque un wait(), entonces el esperará en un pool. Esto implica que la próxima vez que se ejecute un notify() o notifyAll()



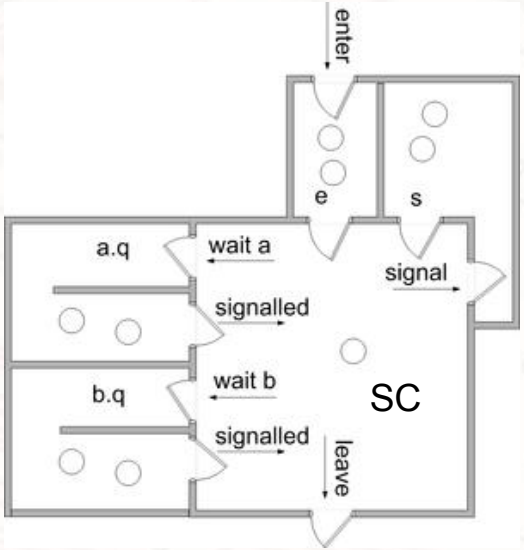
Programando con Hilos

- **Thread**: clase responsable de producir hilos funcionales para otras clase.
- **Runnable**: interfaz que proporciona la capacidad de añadir la funcionalidad de un hilo a una clase simplemente implementando la interfaz.
- Executor y **ExecutorService**: los hilos son mapeados a hilos a nivel del sistema, que son recursos del propio sistema operativo. Si creamos hilos de manera incontrolable, es posible que nos quedemos sin estos recursos rápidamente. El patrón **Thread Pool** ayuda a ahorrar recursos en una aplicación multiproceso/multihilo y a realizar el paralelismo en ciertos límites predefinidos. Esta instancia controla varios subprocesos/hilos reutilizados para ejecutar estas tareas y los va procesando a partir de una cola.
- **Callable** es parecido a Runnable pero con la ventaja de que devuleve un valor. Además podemos usar expresiones Lambda**.**
- **Future es una promesa,** es decir, es el resultado de una **operación asíncrona**. Representa el resultado de una tarea que se completará en el futuro. Es entonces, una vez terminada cuando debemos procesar lo que nos devuelve. Para ello trabajaremos con los métodos get() e isDone()**.**
- Para ejecutar Callables dentro de un Pool o ExecutorService debemos obtener el resultado usando Future.

Sincronización y Comunicación

Sincronización y Comunicación

- Las **variables atómicas** son aquellas que su acceso y modificación se hacen de manera atómica mediante métodos get/set.
- **Cerrojos**, simplemente cierran el paso al recurso compartido hasta que terminan y lo vuelven a abrir. Pueden ser reentrantes (adquirido múltiples veces por el mismo hilo) o de acceso de lectura-escritura y lectura.
- **Semáforos**, variable especial que constituye el método para restringir o permitir el acceso a recursos compartidos.
- Código **sincronizado**, fragmento de código perteneciente a una sección crítica, que asegura que solo pueden tener un hilo ejecutándose dentro de ellos al mismo tiempo.
- **Monitores**, es una estructura de datos que monitoriza la sección crítica. Es el encargado de hacer esperar, notificar a uno o a todos si está libre y asegurarse mediante espera activa que todos los interesados acceden al recurso de manera equitativa y segura, sin producir innanición y de forma exclusiva.

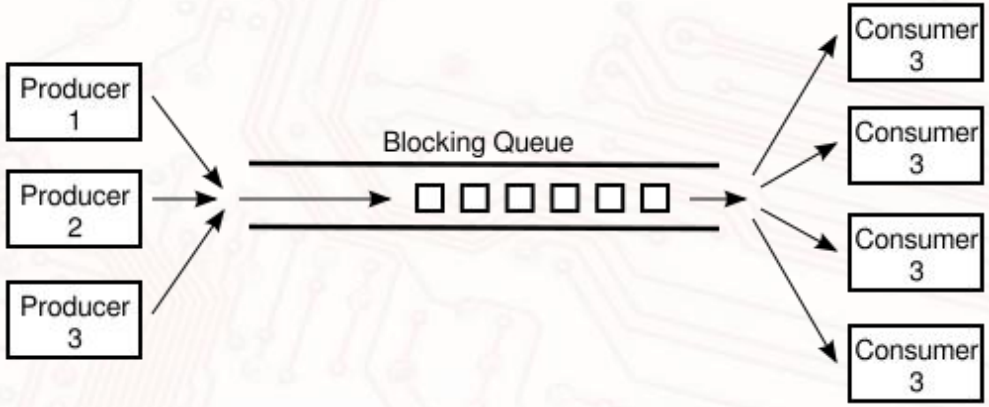


Wikipedia

Ejemplos

Ejemplos: Productor–Consumidor

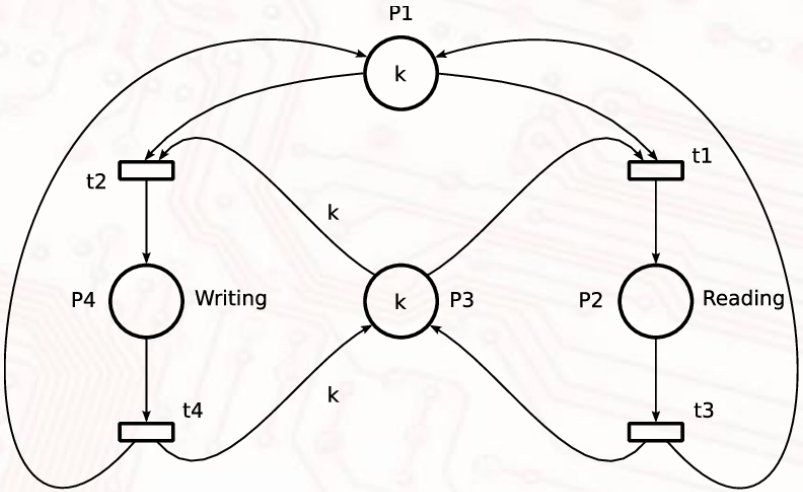
- El programa describe dos procesos, **productor** y **consumidor**, ambos comparten un **buffer de tamaño finito**. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.



Wikipedia

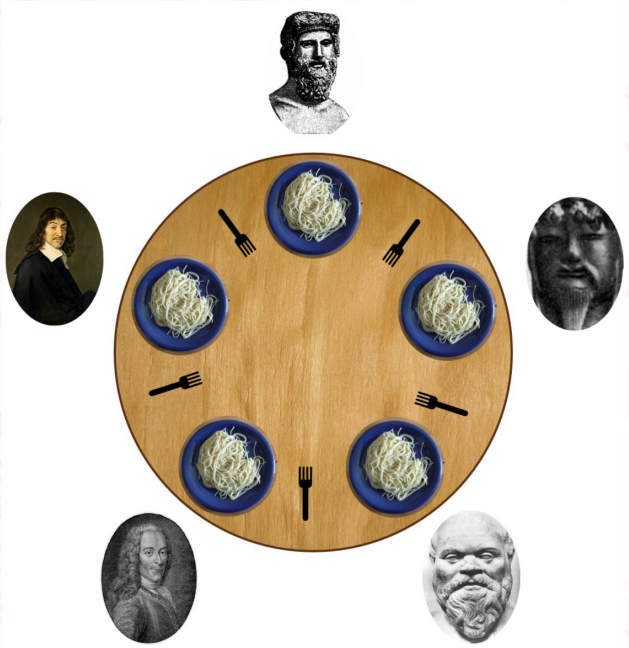
Ejemplos: Lectores-Escritores

- Se produce si varios procesos pueden compartir un archivo o registro de datos. Tenemos varios "**procesos de lectura**" y otros "**procesos de escritura**". Se permite que varios objetos lean un objeto compartido al mismo tiempo pues no hay cambios en el fichero o recurso. Pero un proceso de escritura y otros procesos de lectura no pueden compartir el objeto al mismo tiempo.



Ejemplos: Filósofos comensales

- Cinco **filósofos** se sientan alrededor de una mesa y pasan su vida cenando y pensando. Cada filósofo tiene un plato de fideos y un tenedor a la izquierda de su plato. Para **comer** los fideos son necesarios **dos tenedores** y **cada filósofo sólo puede tomar los que están a su izquierda y derecha**. Si cualquier filósofo toma un tenedor y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.
- El problema consiste en encontrar un algoritmo que permita que los filósofos nunca se mueran de hambre/inanición, ni produzca interbloqueo.



Wikipedia

Conclusiones

¡Vamos con la práctica!

"Si queremos contar líneas de código, no deberíamos referirnos a ellas como líneas producidas, sino como líneas consumidas"

- Edsger Dijkstra

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19