# Actividades Unidad 1

## A. Ejecución paralela de código

Si observamos el siguiente ejemplo de pseudocódigo

```
public class TestClass {
    int x;

    public void testMethod1() {
        for (int i=1; i <= 5; i++) {
            x++;
        }
    }
    public void testMethod2() {
        for (int j=1; j <= 5; j++) {
            x++;
        }
    }
    public void sequential() {
        x = 0;
        testMethod1();
        testMethod2();
        System.out.println(x);
    }
    public void parallel() {
        x = 0;
        // cobegin-coend means that both methods are run simultaneously
        // These sentences doesn't exist in Java. They are used for
        // sample purposes
        cobegin
            testMethod1();
            testMethod2();
        coend
        System.out.println(x);
    }
}
```

**Responde a las siguientes preguntas respecto al código anterior**

¿Qué valor tendrá x tras ejecutar el método sequential?
¿Qué valor tendrá x tras ejecutar el método parallel?

En el método secuencial, x siempre tendrá el valor 10 después de que el programa haya terminado. Esto se debe a que el método `testMethod1` se ejecuta primero, incrementando el valor de x en 5, y luego el método `testMethod2` se ejecuta, incrementando el valor de x en otros 5. Por lo tanto, el valor de x será 10.
En el método paralelo, depende. Dado que la CPU utiliza la concurrencia para procesar el código, un hilo puede

> calcular el valor de x pero no escribir en él, por lo que el valor se pierde hasta que vuelva a tener el foco. Durante ese tiempo, el otro hilo podría escribir en el valor x, pero dicho valor se perderá una vez que el primer hilo continúe siendo procesado. No podemos predecir el orden de las acciones, por lo que el valor de x estará entre 1 y 10.

# B. Concurrencia hardware

Si habéis comprado un procesador hace poco, o estáis al día en cuanto al hardware, sabréis que una de las características de los procesadores es su número de núcleos (4, 8, 16).

Pero además, al número de núcleos lo acompaña otra característica que es el número de hilos o threads, que suele ser el doble que el de núcleos.

**Núcleos vs Threads**

¿Qué implicación tienen los threads de un procesador con respecto a la concurrencia?
¿Si un equipo tiene 8 núcleos / 16 hilos significa eso que puede ejecutar 16 procesos a la vez?

Cuando hablamos de un núcleo, estamos hablando técnicamente de un procesador; una unidad que puede realizar operaciones matemáticas en un corto período de tiempo. Este procesador no puede hacer más de 1 operación al mismo tiempo, por lo que los fabricantes ponen un grupo de estos ahora llamados núcleos juntos en un solo microchip, este último siendo nombrado el procesador. Este es un núcleo: una unidad que resuelve operaciones.
Sin embargo, los hilos que aparecen junto a ellos (generalmente siendo el doble que los núcleos) se deben a una tecnología llamada multihilo simultáneo. Esto permite al procesador, mientras espera a que se complete una instrucción, ejecutar una segunda tarea; aprovechando los momentos en los que el núcleo no está haciendo nada, aunque no es tan efectivo como tener más núcleos, ya que no duplica el rendimiento, sino más bien un 50% aproximadamente.
Aún así, si miramos el administrador de tareas, podemos ver más de 16 tareas ejecutándose simultáneamente. Esto se debe a que los hilos también son una unidad lógica (es decir, no existe físicamente en el procesador), compuesta por el sistema operativo, donde las tareas se cambian constantemente, en una unidad de tiempo tan mínima que parece estar computando dos acciones al mismo tiempo. Es por eso que, aunque diga que su procesador tiene, por ejemplo, 4 núcleos y 8 hilos, puede ejecutar incluso cientos de procesos simultáneamente. NO en paralelo. El paralelismo real solo ocurre uno por cada núcleo; todo lo relacionado con los hilos es para aumentar el rendimiento de su máquina.
En resumen, tener 8 núcleos y 16 hilos significa que el procesador puede ejecutar 16 hilos de proceso simultáneamente, pero no necesariamente 16 procesos diferentes. Los hilos son unidades lógicas que permiten que el procesador ejecute múltiples tareas de manera más eficiente, pero no necesariamente al mismo tiempo.

# C. Escalado de sistemas

La capacidad de procesamiento de los sistemas se puede ampliar de diferentes formas. Todo va a depender de si estamos trabajando con sistemas on-premise o con sistemas en la nube.

Con escalado nos referimos a la posibilidad de incrementar las capacidades de un sistema.

**Escalado horizontal vs vertical**

Explica brevemente qué es el escalado horizontal y qué es el escalado vertical.
Investiga las diferencias, ventajas e inconvenientes del escalado horizontal y el escalado vertical.
¿A qué tipo de sistemas (on-premise vs cloud) se aplica cada uno de ellos?

Llamamos escalado horizontal cuando agregamos más nodos a nuestro sistema, es decir, integrando nuevas máquinas (probablemente, para trabajar como un servidor) para satisfacer la alta demanda que podríamos estar sufriendo.
Por otro lado, el escalado vertical consiste en incrementar la potencia de nuestra máquina actual para satisfacer la demanda; podría ser aumentando la cantidad de memoria RAM, agregando más discos duros, o cambiando el procesador actual por otro más potente.
Las principales diferencias entre el escalado horizontal y vertical son los recursos. Mientras que el último es más fácil de implementar, ya que solo necesitamos actualizar nuestra máquina actual, también es muy ineficiente en costos a largo plazo. Además, estamos limitados por la cantidad de capacidad que una sola máquina puede manejar.
El escalado horizontal requiere más dinero, esfuerzo y puede implicar un aumento en la complejidad. Sin embargo, este modelo no se basa en una sola máquina para manejar todas las peticiones; de hecho, algunos nodos podrían incluso caer y el sistema puede funcionar normalmente mientras se reemplazan.
Podemos determinar que, en función del tipo de sistema con el que esté trabajando, puede elegir entre 2 opciones. Si está tratando con un sistema en las instalaciones del cliente (on-premise), es más aconsejable un escalado vertical, llegando este a suponer en ocasiones el reemplazo de un equipo antiguo por uno más moderno cuando se ha llegado al límite de ampliación.
Por otro lado, si estamos en un sistema en la nube, donde los recursos pueden, o parecen, ser limitados, el escalado vertical es la mejor opción. Por lo tanto, si lo que necesita es un alto rendimiento, un diseño robusto y un sistema estable a largo plazo (y, por supuesto, mucho dinero detrás), debería optar por el escalado horizontal.

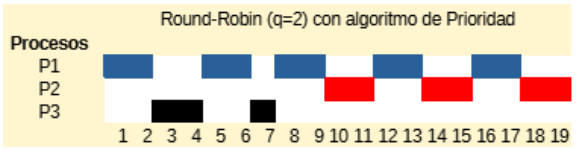
# D. Planificación de procesos

Dados los tres procesos que se ponen como ejemplo en la unidad

| Proceso | Tiempo de llegada | Tiempo uso CPU | Prioridad |
|---|---|---|---|
| P1 | 0 | 10 | 5 |
| P2 | 1 | 6 | 10 |
| P3 | 2 | 3 | 7 |

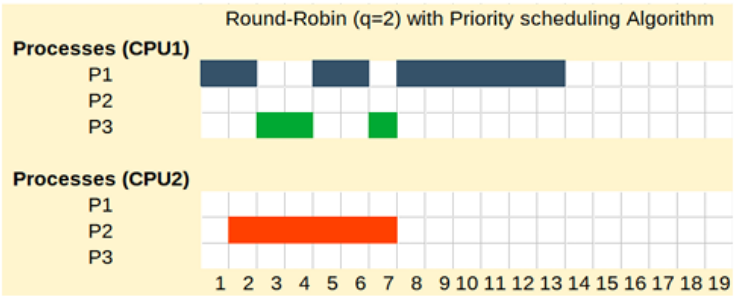
**Planificador combinado**

En realidad, nunca se usa una única estrategia de planificación, sino que lo más común es que se combinen varias de ellas. De hecho en Round-Robin hemos usado también FCFS.
¿Te atreves a ver cómo sería una planificación Round-Robin con prioridad, con un único procesador? Ten en cuenta que funcionará con el quantum y a la hora de escoger el siguiente proceso a ejecutar, se basará en la prioridad de los que haya en la lista.
¿Y la misma planificación con 2 procesadores?




| | $T_{waiting}$ | $T_{turnaround}$ | $CPU_{Usage}$ | Throughput |
|---|---|---|---|---|
| P1 | 7 | 17 | | |
| P2 | 12 | 18 | | |
| P3 | 2 | 5 | | |
| | | | 100,00 % | 3/19=0,15 |

Round-Robin with Priority Algorithm Q = 2 (2 processor system):




| | $T_{WAITING}$ | $T_{tuRNArouND}$ | $Usage_{CPU}$ | Throughput |
|---|---|---|---|---|
| P1 | 3 | 13 | | |
| P2 | 0 | 6 | | |
| P3 | 2 | 5 | 19/26 | 3/13 |
| | | | 73,00 % | 23,00 % |

# E. Condiciones de Bernstein

Dadas las siguientes instrucciones de código, queremos saber cuáles pueden ejecutarse de forma simultánea sin tener porblemas de concurrencia.

```
1. cuad := x * x;
2. m1 := a * cuad;
3. m2 := b * x;
4. z := m1 + m2;
5. y := z + c;
```

**Condiciones de Bernstein**

Aplica las condiciones de Bernstein para identificar qué instrucciones de las siguientes pueden ser ejecutadas de forma simultánea sin problemas de concurrencia.
Indicar, para cada par de instrucciones, si son independientes, o están acopladas.

| NUMERO INSTRUCCIÓN | | Conjunto L | Conjunto E |
|---|---|---|---|
| 1 | **cuad := x * x;** | L(S1) = {x} | E(S1) = {cuad} |
| 2 | **m1 := a * cuad;** | L(S2) = {a, cuad} | E(S2) = {m1} |
| 3 | **m2 := b * x;** | L(S3) = {b, x} | E(S3) = {m2} |
| 4 | **z := m1 + m2;** | L(S4) = {m1, m2} | E(S4) = {z} |
| 5 | **y := z + c;** | L(S5) = {z, c} | E(S5) = {y} |

| | | |
|---|---|---|
| NO | $L(S_1) \cap E(S_2) = \emptyset$ $E(S_1) \cap L(S_2) = \{cuad\} \neq \emptyset$ $E(S_1) \cap E(S_2) = \emptyset$ | Instrucciones 1 y 2 NO se pueden ejecutar concurrentemente |
| SI | $L(S_1) \cap E(S_3) = \emptyset$ $E(S_1) \cap L(S_3) = \emptyset$ $E(S_1) \cap E(S_3) = \emptyset$ | Instrucciones 1 y 3 SI se pueden ejecutar concurrentemente |
| SI | $L(S_1) \cap E(S_4) = \emptyset$ $E(S_1) \cap L(S_4) = \emptyset$ $E(S_1) \cap E(S_4) = \emptyset$ | Instrucciones 1 y 4 SI se pueden ejecutar concurrentemente |
| SI | $L(S_1) \cap E(S_5) = \emptyset$ $E(S_1) \cap L(S_5) = \emptyset$ $E(S_1) \cap E(S_5) = \emptyset$ | Instrucciones 1 y 5 SI se pueden ejecutar concurrentemente |
| SI | $L(S_2) \cap E(S_3) = \emptyset$ $E(S_2) \cap L(S_3) = \emptyset$ $E(S_2) \cap E(S_3) = \emptyset$ | Instrucciones 2 y 3 SI se pueden ejecutar concurrentemente |
| NO | $L(S_2) \cap E(S_4) = \emptyset$ $E(S_2) \cap L(S_4) = \{m1\} \neq \emptyset$ $E(S_2) \cap E(S_4) = \emptyset$ | Instrucciones 2 y 4 NO se pueden ejecutar concurrentemente |
| SI | $L(S_2) \cap E(S_5) = \emptyset$ $E(S_2) \cap L(S_5) = \emptyset$ $E(S_2) \cap E(S_5) = \emptyset$ | Instrucciones 2 y 5 SI se pueden ejecutar concurrentemente |
| NO | $L(S_3) \cap E(S_4) = \emptyset$ $E(S_3) \cap L(S_4) = \{m2\} \neq \emptyset$ $E(S_3) \cap E(S_4) = \emptyset$ | Instrucciones 3 y 4 NO se pueden ejecutar concurrentemente |
| SI | $L(S_3) \cap E(S_5) = \emptyset$ $E(S_3) \cap L(S_5) = \emptyset$ $E(S_3) \cap E(S_5) = \emptyset$ | Instrucciones 3 y 5 SI se pueden ejecutar concurrentemente |
| NO | $L(S_4) \cap E(S_5) = \emptyset$ $E(S_4) \cap L(S_5) = \{z\} \neq \emptyset$ $E(S_4) \cap E(S_5) = \emptyset$ | Instrucciones 4 y 5 NO se pueden ejecutar concurrentemente |