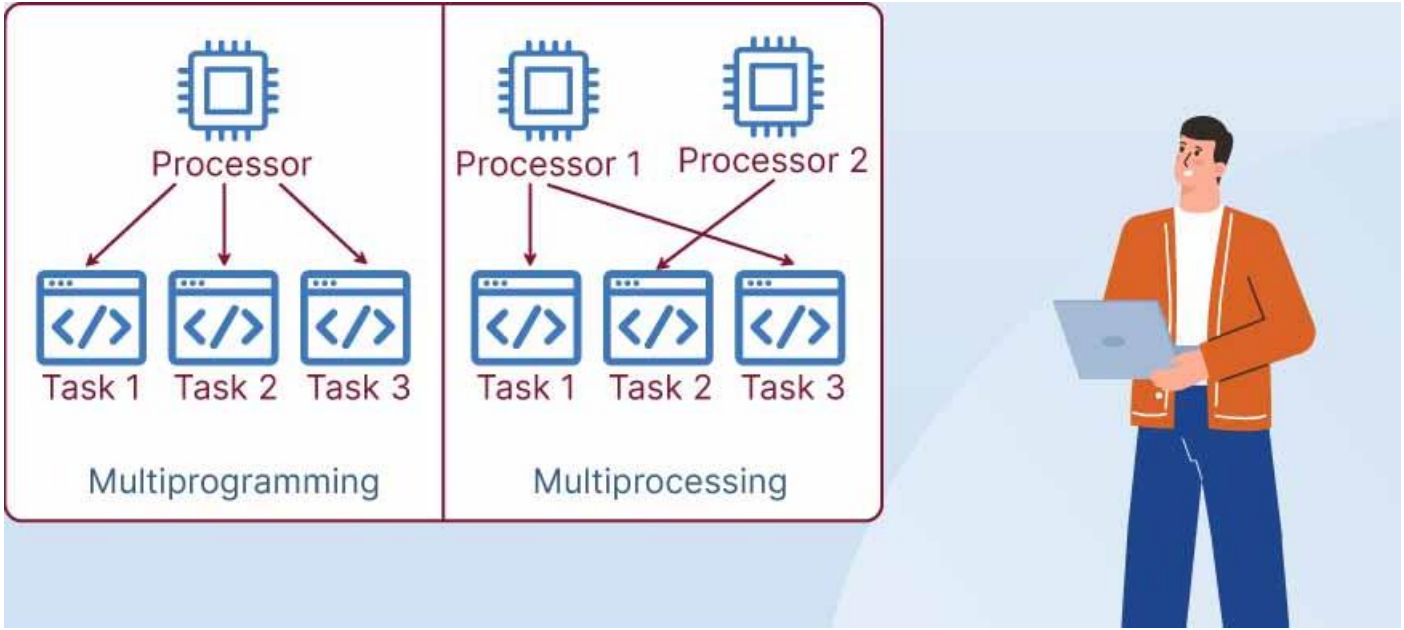


Multiproceso

Índice

1. Ejecutable

Ejecutable: Archivo con la estructura necesaria para que el sistema operativo pueda poner en marcha el programa que hay dentro.

Aplicación: Un tipo de programa informático diseñado como herramienta para resolver de manera automática un problema específico del usuario. Al instalarla en un equipo observamos que puede estar formada por varios ejecutables o librerías. Siempre que lancemos la ejecución de una aplicación se creará al menos un proceso nuevo en nuestro sistema.

1. Procesos

Proceso: Es un programa en ejecución. Archivo que está ejecutándose y bajo el control del sistema operativo.

Bloque de Control de Proceso (BCP): estructura de datos donde se almacena información acerca de un proceso:

- Identificación del proceso
- Estado del proceso
- Contador de programa
- Registros de CPU
- Información de planificación de CPU con la prioridad
- Información de gestión de memoria
- Información contable como la cantidad de tiempo de CPU y tiempo real consumido
- Información de estado E/S, lista de dispositivos asignados, archivos, etc.

1. Servicios

Servicio: Proceso que no muestra ninguna ventana ni gráfico en pantalla porque no está pensado para que el usuario lo maneje directamente.

Es un programa que atiende a otro programa.

1. Ejercicio

Crea un programa simple en Java y realiza la ejecución del mismo desde el IDE. Localiza el proceso que se ha lanzado en el sistema y observa su comportamiento.

Realiza modificaciones en el programa para que perturbe su consumo y la utilización de los recursos.

1. Tipos de ejecución de procesos

Tres tipos:

- **Ejecución por lotes**: El usuario solo está interesado en el resultado final y no le interesa interactuar con la aplicación. Introduce datos de entrada y permanece a la espera hasta que finalice.
- **Interactivos**: El proceso interactúa continuamente con el usuario, indicándole los pasos que debe seguir para completar la tarea. Por ejemplo cualquier aplicación con GUI, procesadores de texto.
- **Tiempo real**: Tareas en las que es crítico el tiempo de respuesta por parte del proceso.
  Por ejemplo el ordenador de a bordo de un coche en que controla los obstáculos que se encuentra en la carretera.

1. Ejecución con argumentos de entrada

- Los programas pueden ejecutarse sin valores de entrada o por el contrario se pueden ejecutar con unos valores o argumentos iniciales.

- Estos valores deben ser indicados al programa en el momento previo a su ejecución

- El programa hará uso de dichos argumentos de entrada.

1. Ejercicio

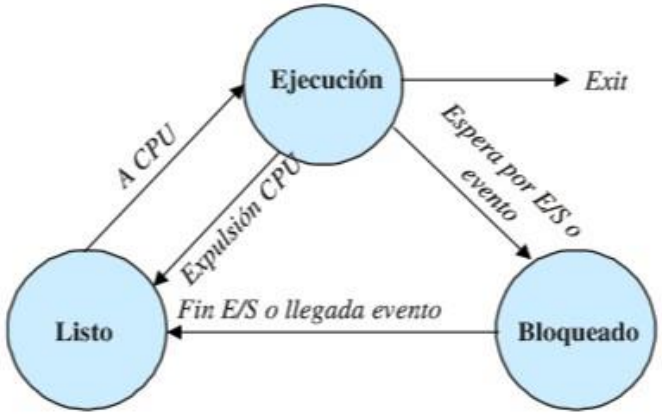
Crea un programa en Java que para poder ser ejecutado deba recibir tres argumentos de entrada. Dichos argumentos serán números. El programa deberá devolver por pantalla el mayor de los tres números introducidos.

1. Estado de los procesos

Ciclo de vida o estados:

- **En ejecución**: está dentro del microprocesador.
- **Bloqueado**: el proceso no puede hacer nada hasta que no ocurra un evento externo.
- **Listo**: el proceso está parado temporalmente y listo para ejecutarse cuando se le de oportunidad.

1. Estado de los procesos



- **En ejecución - Bloqueado**: espera la ocurrencia de un evento externo.
- **Bloqueado - Listo**: ocurre el evento externo que esperaba.
- **Listo – En ejecución**: el sistema le otorga un tiempo de CPU.
- **En ejecución – Listo**: acaba el tiempo asignado por el sistema operativo.

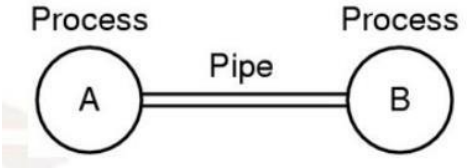
1. Procesos - Comunicación

Las operaciones multiproceso pueden implicar que sea necesario comunicar información entre muchos procesos, lo que obliga a la necesidad de utilizar mecanismos específicos de comunicación que ofrecerá Java o a diseñar alguno separado que evite los problemas que puedan aparecer.

1. Procesos - Gestión

PIPES SIN NOMBRE

Falso fichero que sirve para conectar dos procesos.



Si el proceso A quiere enviar datos al proceso B, los escribe en el pipe como si fuera un fichero de salida.
Si la pipe está llena, se bloqueará hasta que se vacíe.

El proceso B puede leer los datos sin más que leer el pipe como si se tratara de un fichero de entrada.
Si la pipe está vacía, se bloqueará hasta que algún proceso ponga datos en él.

1. Procesos – Algoritmos de planificación

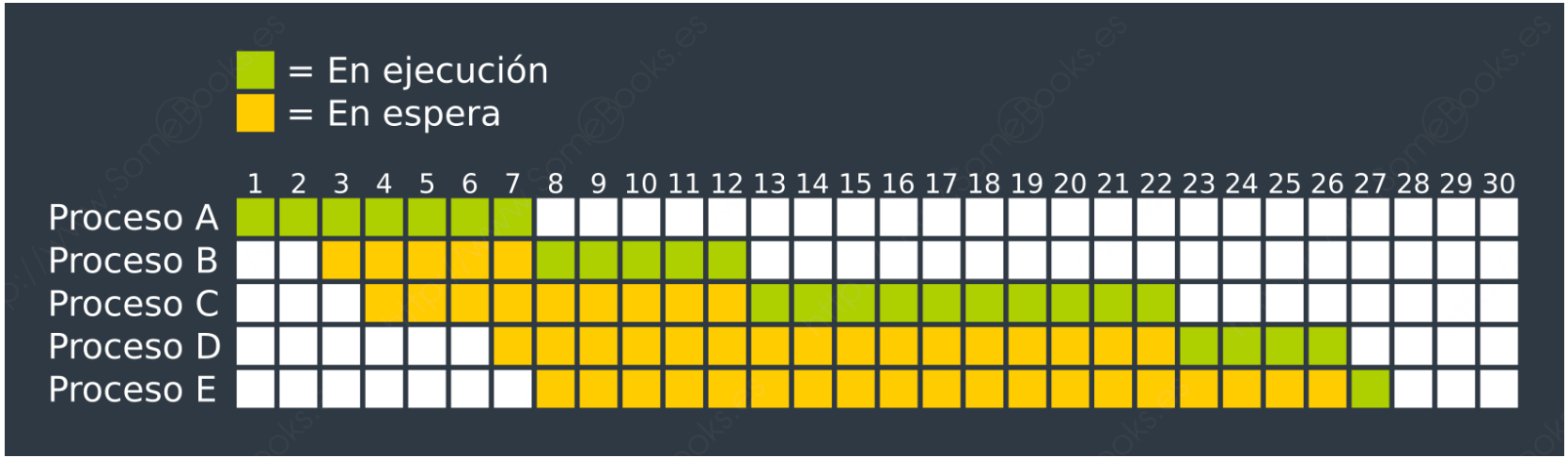
Indican la forma en que el tiempo de procesamiento debe repartirse entre todas las tareas que deben ejecutarse en un momento determinado.

- FCFS
- RR
- SPF
- SRT
- Varias colas con realimentación

1. Procesos – Algoritmos de planificación

FCFS (First Come First Served)

El primer proceso que llegue al procesador se ejecuta antes y de forma completa. Hasta que su ejecución no termina no podrá pasarse a ejecutar otro proceso.

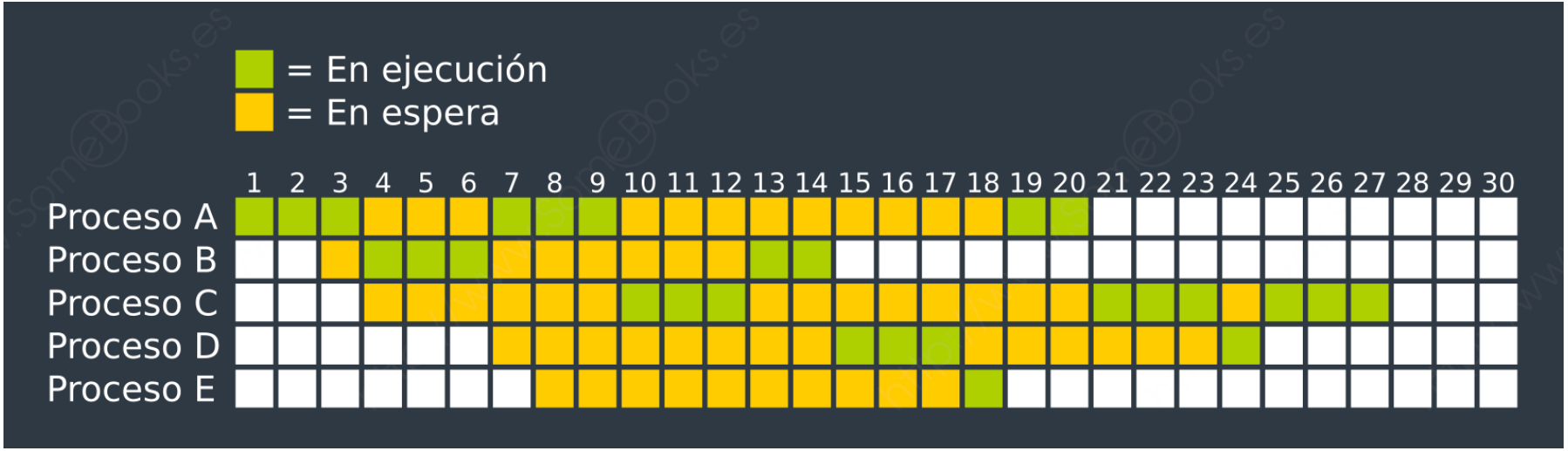


1. Procesos – Algoritmos de planificación

RR (Round Robin) – Turno rotatorio

Se designa una cantidad corta de tiempo (quantum) de procesamiento a todas las tareas.

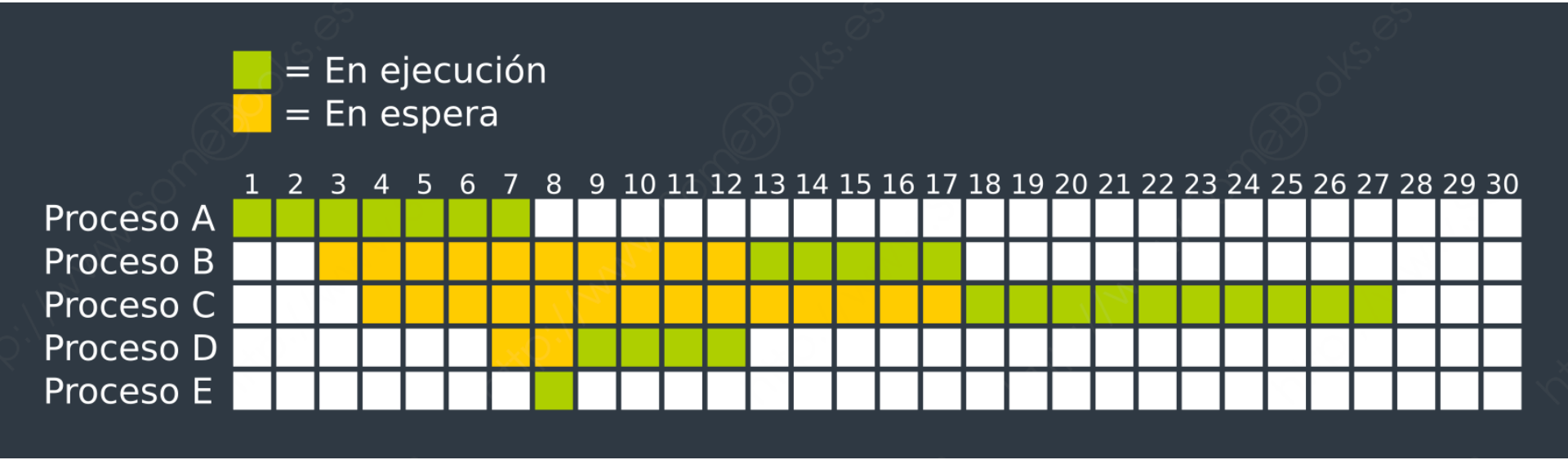
Las que necesiten más tiempo de proceso deberán esperar a que vuelva a ser su turno para seguir ejecutándose.



1. Procesos – Algoritmos de planificación

SPF (Shortest Process First)

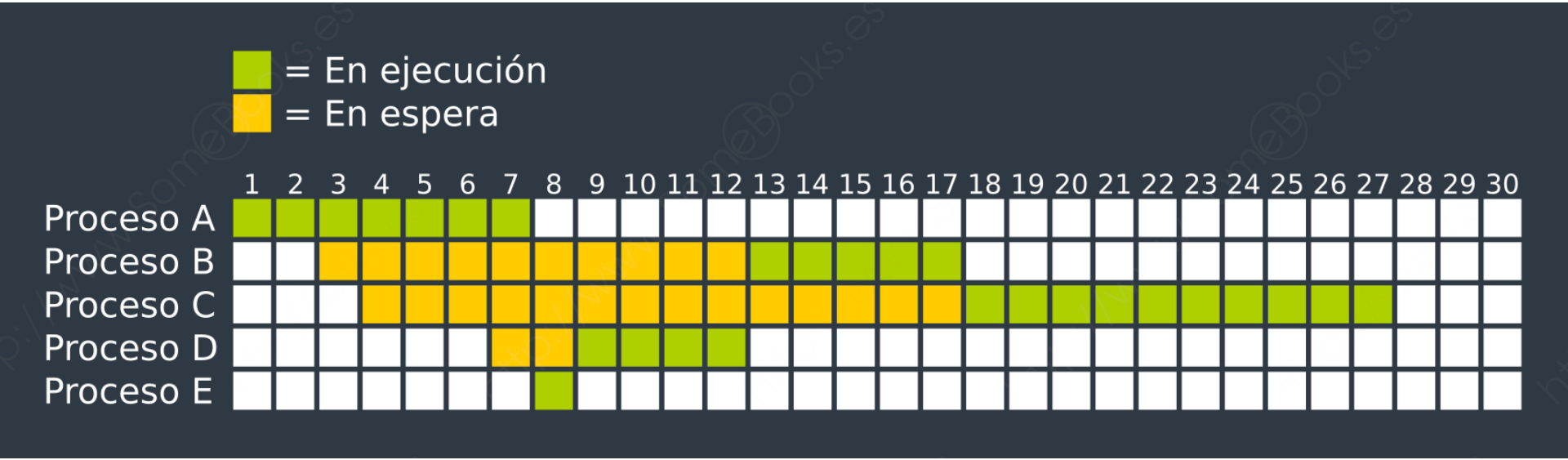
De entre todos los procesos listos para ser ejecutados, lo hará primero el más corto.



1. Procesos – Algoritmos de planificación

SRT (Shortest Remaining Time)

De entre todos los procesos listos para ser ejecutados, lo hará primero el que le quede menos tiempo para terminar.



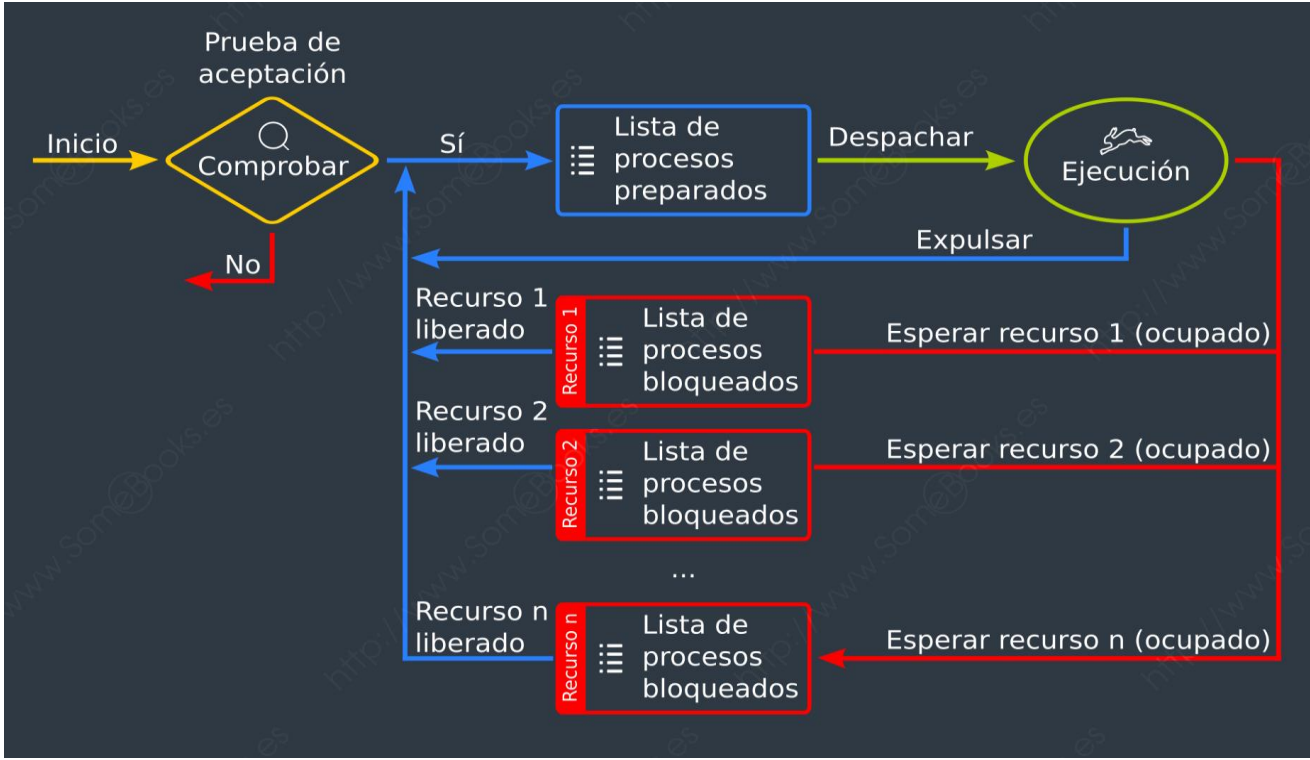
1. Procesos – Algoritmos de planificación

Varias colas con realimentación

Se conoce el tiempo de ejecución del proceso al inicio.

El sistema dispone de varias colas que a su vez pueden disponer de diferentes políticas.

Los procesos van pasando de una cola a otra hasta que terminan su ejecución.



2. Multiproceso vs. Multihilo

Multiproceso

El multiproceso consiste en la ejecución de varios procesos diferentes de forma *simultánea* para la realización de una o varias tareas relacionadas o no entre sí. En este caso, cada uno de estos procesos es una aplicación independiente. El caso más conocido es aquel en el que nos referimos al Sistema Operativo (Windows, Linux, MacOS, . . .) y decimos que es multitarea puesto que es capaz de ejecutar varias tareas o procesos (o programas) *al mismo tiempo*

2. Multiproceso vs. Multihilo

Multihilo

Hablamos de multihilo cuando se ejecutan varias tareas relacionadas o no entre sí dentro de una misma aplicación. En este caso no son procesos diferentes sino que dichas tareas se ejecutan dentro del mismo proceso del Sistema Operativo. A cada una de estas tareas se le conoce como hilo o thread (en algunos contextos también como procesos ligeros)

2. Multiproceso vs. Multihilo

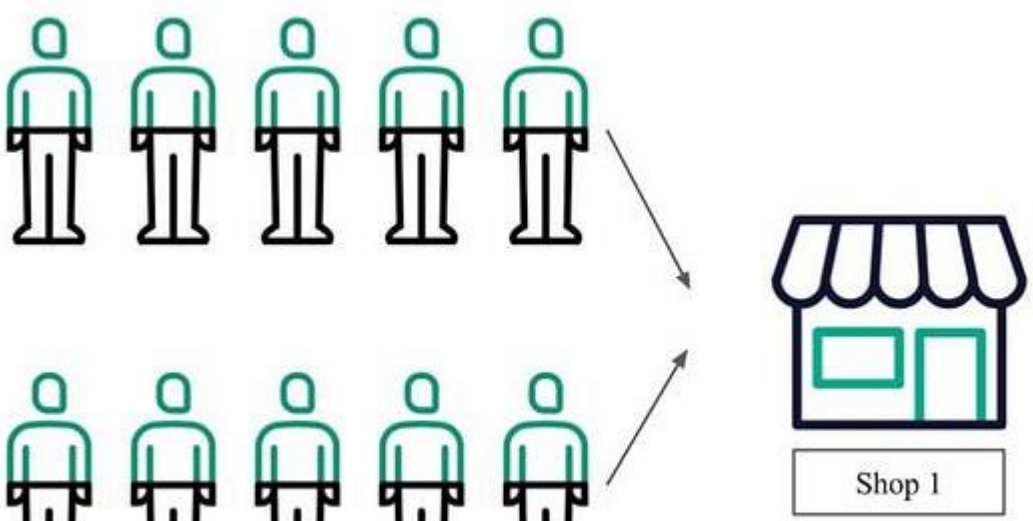
Ejecución por el Sistema Operativo

- Tanto en Multiproceso como en Multihilo estaríamos hablando de lo que se conoce como Programación Concurrente.
- Hay que tener en cuenta que en ninguno de los dos casos la ejecución es realmente simultánea (o no necesariamente), ya que el Sistema Operativo es quién hace que parezca así, pero los ejecuta siguiendo los algoritmos de planificación.

3. Programación concurrente, paralela y distribuida

Programación Concurrente

Es la programación de aplicaciones capaces de realizar varias tareas de forma simultánea utilizando hilos o threads. En este caso todas las tareas compiten por el uso del procesador y en un instante determinado sólo una de ellas se encuentra en ejecución. Además, habrá que tener en cuenta que diferentes hilos pueden compartir información entre sí y eso complica mucho su programación y coordinación.

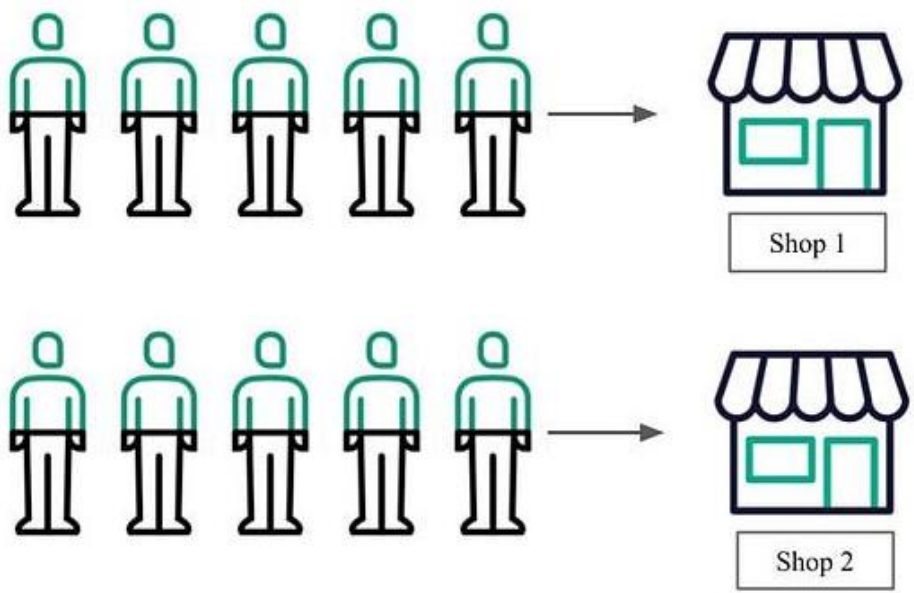


3. Programación concurrente, paralela y distribuida

Programación Paralela

Es la programación de aplicaciones que ejecutan tareas de forma paralela, de forma que no compiten por el procesador puesto que cada una de ellas se ejecuta en uno diferente.

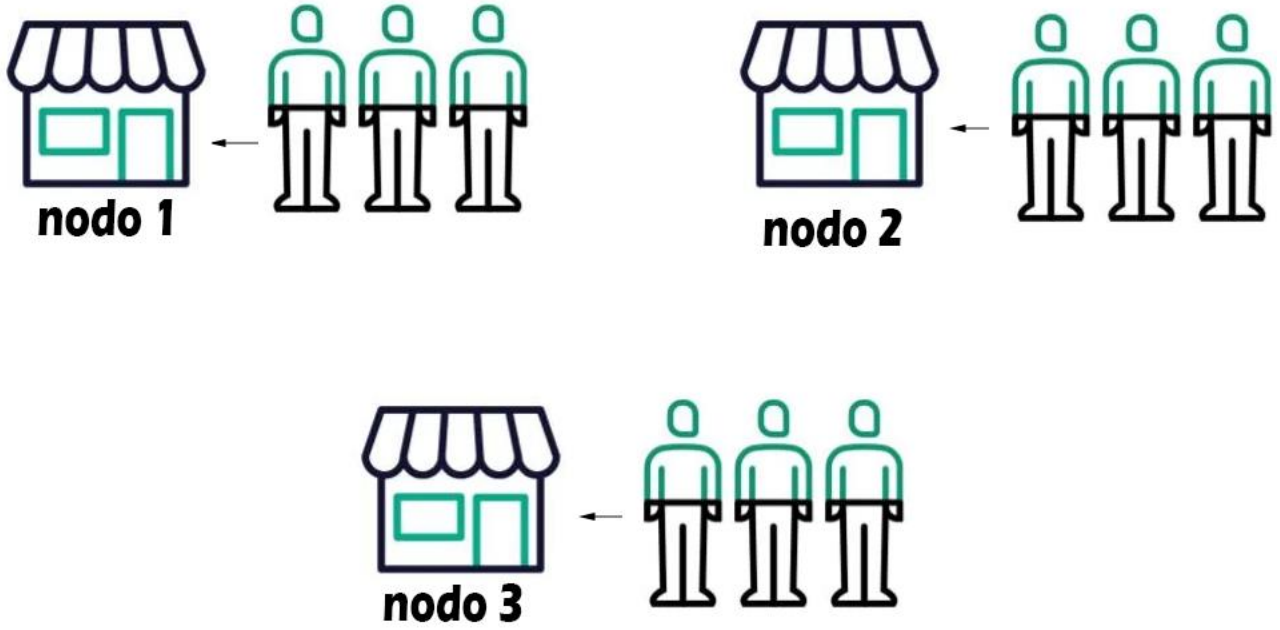
Normalmente buscan resultados comunes dividiendo el problema en varias tareas que se ejecutan al mismo tiempo.



3. Programación concurrente, paralela y distribuida

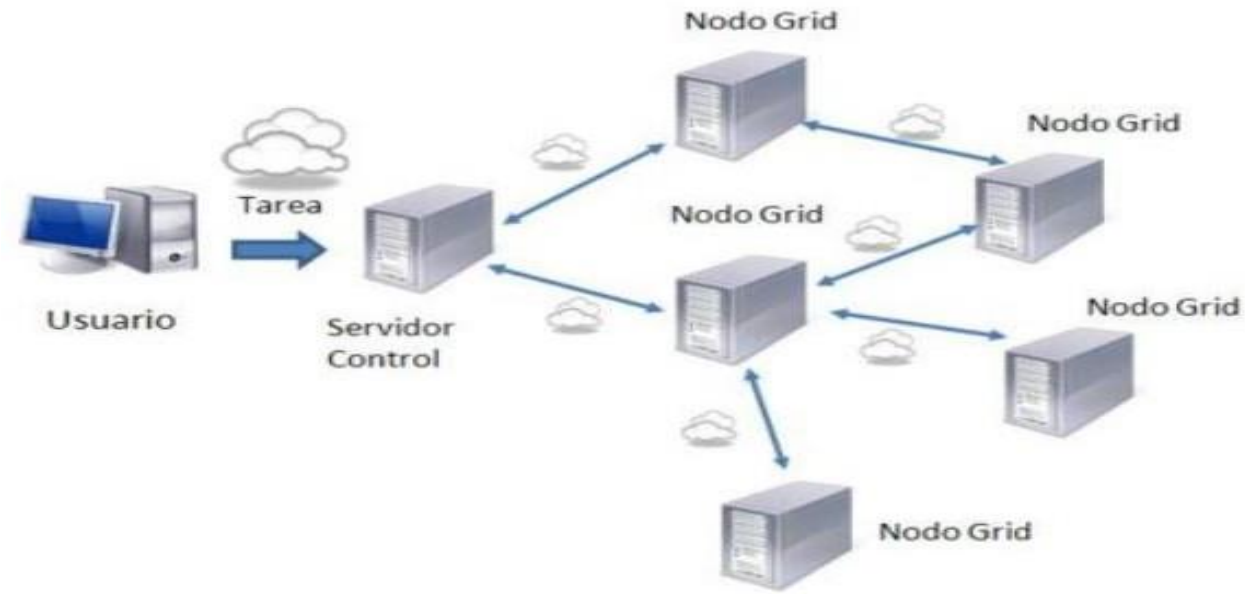
Programación Distribuida

Es la programación de aplicaciones en las que las tareas a ejecutar se reparten entre varios equipos diferentes (conectados en red a los que llamaremos nodos). Juntos, estos equipos, forman lo que se conoce como un Sistema Distribuido, que busca formar redes de equipos que trabajen con un fin común



3. Programación concurrente, paralela y distribuida

Programación Distribuida



4. Creación de procesos

Procesos – Creación en Java

Java permite la creación de procesos, desde un programa. Para ello dispone de varias clases para esta gestión:

- Clase java.lang.ProcessBuilder
- Clase java.lang.RunTime

4. Creación de procesos

ProcessBuilder

```
ProcessBuilder pb;
try {
  pb = new ProcessBuilder();
  pb.start();
} catch (Exception e) {
  logger.log(Level.SEVERE, "Error al crear el ProcessBuilder", e);
}
```

4. Creación de procesos

RunTime

```
String[] comandos = new String[1];
comandos[0] = (ruta);
try {
  Runtime rt = Runtime.getRuntime();
  rt.exec(comandos);
} catch (Exception e) {
  logger.log(Level.SEVERE, "Error ejecutando el proceso", e);
}
```

4. Creación de procesos

Ejemplo de clase Lanzador

```
public class LanzaProcesos {
  private static final Logger logger = Logger.getLogger(LanzaProcesos.class.getName());

  public void ejecutar(String ruta) {
    ProcessBuilder pb;
    try {
      pb = new ProcessBuilder(ruta);
      pb.start();
    } catch (Exception e) {
      logger.log(Level.SEVERE, "Error executing process", e);
    }
  }

 public static void main(String[] args) {
    String ruta = "C:\\Program Files (x86)\\Adobe\\Acrobat Reader DC\\Reader\\AcroRd32.exe";

    LanzaProcesos lp = new LanzaProcesos();
    lp.ejecutar(ruta);
    System.out.println("Finalizado");

  }
}
```

4. Creación de procesos

Añadir parámetros a los procesos

Es posible añadir parámetros a los procesos que lancemos desde nuestro programa en Java. Por ejemplo, podemos indicar que:

- Cuando se lance un programa determinado lo haga abriendo un fichero en concreto
- Cuando se abra un navegador lo haga por defecto con la página indicada

4. Creación de procesos

Añadir parámetros con ProcessBuilder

```
pb = new ProcessBuilder();
pb.command().add(param1);
pb.command().add(param2);
pb.start();
```

4. Creación de procesos

Añadir parámetros con RunTime

```
String[] comandos = new String[2];
comandos[0] = (param1);
comandos[1] = (param2);
try {
  Runtime rt = Runtime.getRuntime();
  rt.exec(comandos);
} catch (Exception e) {
  logger.log(Level.SEVERE, "Error executing command", e);
}
```

END

jgardur081@g.educaand.es