



2ºDAM - Programación de servicios y procesos

Tema I - Introducción

ÍNDICE

- 1. Repaso programación Java
 - a) Utilizar los sangrados correctamente
 - b) Comentar de manera correcta el código
 - c) Nombrar las variables de forma estandarizada
- 2. Definición y estructura de programa y proceso
 - a) Fichero ejecutable
 - b) Proceso
- 3. Estados de un proceso en Linux
- 4. Diagrama de estados de un proceso

ÍNDICE

1. Repaso programación Java

- a) Utilizar los sangrados correctamente
- b) Comentar de manera correcta el código
- c) Nombrar las variables de forma estandarizada

2. Definición y estructura de programa y proceso

- a) Fichero ejecutable
- b) Proceso

3. Estados de un proceso en Linux

4. Diagrama de estados de un proceso

1. Repaso programación Java

La tecnología Java es:

- **Lenguaje de programación Java** es un lenguaje de alto nivel, orientado a objetos. El lenguaje es inusual porque los programas Java son tanto **compilados** como **interpretados**. La compilación traduce el código Java a un lenguaje intermedio llamado Java bytecode. El **Bytecode**, es analizado y ejecutado (interpretado) por **Java Virtual Machine (JVM)** — un traductor entre bytecode, el sistema operativo subyacente y el hardware. Todas las implementaciones del lenguaje de programación deben emular JVM, para permitir que los programas Java se ejecuten en cualquier sistema que tenga una versión de JVM.

1. Repaso programación Java

La tecnología Java es:

- La **plataforma Java** es una plataforma sólo de software que se ejecuta sobre varias plataformas de hardware. Está compuesto por **JVM** y la **interfaz de programación de aplicaciones (API) Java** — un amplio conjunto de componentes de software (clases) listos para usar que facilitan el desarrollo y despliegue de applets y aplicaciones. La API Java abarca desde objetos básicos a conexión en red, seguridad, generación de XML y servicios web. Está agrupada en bibliotecas — conocidas como **paquetes** — de clases e interfaces relacionadas.

1. Repaso programación Java

Versiones de la plataforma:

- **Java SE** (Plataforma Java, Standard Edition). Permite desarrollar y desplegar aplicaciones Java en desktops y servidores, como también en entornos empotrados y en tiempo real.
- **Java EE** (Plataforma Java, Enterprise Edition). La versión empresarial ayuda a desarrollar y desplegar aplicaciones Java en el servidor portables, robustas, escalables y seguras.
- **Java ME** (Plataforma Java, Micro Edition). Proporciona un entorno para aplicaciones que operan en una gama amplia de dispositivos móviles y empotrados, como teléfonos móviles, PDA, STB de TV e impresoras.

Para la implementación y desarrollo de aplicaciones, nos servimos de un **IDE** (Entorno Integrado de Desarrollo), que es un programa informático formado por distintas herramientas de programación; como son: editor, compilador, intérprete, depurador, etc.

1. Repaso programación Java

Vamos a repasar los siguientes conceptos mediante la realización de ejercicios de Java utilizando el IDE de Eclipse:

1. Clases
2. Atributos
3. Constructores
4. Métodos
5. Bucles
6. Nivel de acceso

Repaso normas de codificación:

http://javafoundations.blogspot.com/2010/07/java-estandares-de-programacion.html#1_5_4_declaracion_clases_interfaces

Más información sobre Java: [Oracle Academy](#)

1. Repaso programación Java

- 1. **Clases:** estructura básica de una clase. Crea una clase java que se llame Coche

Estructura de una Clase

```
acceso class nombre de la clase
```

```
{
```

```
// atributos
```

```
acceso tipo variable-1 ;
```

```
acceso tipo variable-2;
```

```
....
```

```
acceso tipo de variable-n;
```

```
//métodos
```

```
acceso tipo nombre_metodo1(lista de parámetros){
```

```
cuerpo del metodo
```

```
}
```

```
....
```

```
acceso tipo nombre_metodo2(lista de parámetros){
```

```
cuerpo del metodo
```

```
}
```

```
} //fin de la clase
```

← Atributos: Campos (datos)

← Métodos: Comportamiento (procedimientos)

1. Repaso programación Java

2. **Atributos:** incluye en la clase los siguientes atributos: numPuertas, color, matricula, modelo, anioMatriculacion y km. Utiliza un tipo de datos adecuado para cada atributo.

3. **Constructores.** Crea un constructor que inicialice los atributos de la clase Coche.

4. **Métodos:** crea los métodos get y set para poder acceder a la lectura/escritura de los atributos de la clase.

5. **Bucles:** crea un método que calcule cuántos kilómetros realiza el coche de media en un año y lo muestre por pantalla. Para ello divide el número de kilómetros que tiene el coche entre el número de años que tiene. Puedes calcular cuántos años tiene el coche desde el año de matriculación del coche al año actual (2022). En el caso de que el coche tenga menos de un año se mostrará por pantalla el mensaje siguiente "Coche nuevo, no tiene suficiente antigüedad".

1. Repaso programación Java

6. Nivel de acceso. Revisa los atributos y métodos para indicar el nivel de acceso deseado. En este caso queremos poder acceder a la información de la clase Coche desde la propia clase.

Modificador de acceso	Public	Protected	Private	Sin especificar (package)
¿El método o atributo es accesible desde la propia clase?	Sí	Sí	Sí	Sí
¿El método o atributo es accesible desde otras clases en el mismo paquete?	Sí	Sí	No	Sí
¿El método o atributo es accesible desde una subclase en el mismo paquete?	Sí	Sí	No	Sí
¿El método o atributo es accesible desde subclases en otros paquetes?	Sí	Sí	No	No
¿El método o atributo es accesible desde otras clases en otros paquetes?	Sí	No	No	No

1. Repaso programación Java

- 7. Recuerda **herencia y polimorfismo**. En Java podemos heredar de una sola clase pero podemos implementar de N interfaces. Estas interfaces engloban comportamientos (métodos) que después tendremos que implementar en nuestras clases.

Crea una jerarquía de clases a partir de una llamada Vehículo que sea la clase Padre.

Después, debes implementar la interfaz Movimientos con los métodos siguientes donde corresponda:

- arrancar(): este método indica si el vehículo está o no arrancado.
- correr(double km): este método pone en movimiento al vehículo y le suma los km al total de km que ya tenía el vehículo antes de correr.
- parar(): este método para el vehículo y cambia el estado arrancado para indicar que ya no lo está.

ÍNDICE

1. Repaso programación Java

- a) Utilizar los sangrados correctamente
- b) Comentar de manera correcta el código
- c) Nombrar las variables de forma estandarizada

2. Definición y estructura de programa y proceso

a) Fichero ejecutable

b) Proceso

3. Estados de un proceso en Linux

4. Diagrama de estados de un proceso

2. Definición y estructura de programa y proceso

¿Es lo mismo una aplicación, un ejecutable y un proceso?

Una aplicación es un tipo de programa informático, diseñado como herramienta para resolver de manera automática un problema específico del usuario.

Debemos darnos cuenta de que sobre el hardware del equipo, todo lo que se ejecuta son programas informáticos, que, ya sabemos, que se llama software. Con la definición de aplicación anterior, buscamos diferenciar las aplicaciones, de otro tipo de programas informáticos, como pueden ser: los sistemas operativos, las utilidades para el mantenimiento del sistema, o las herramientas para el desarrollo de software. Por lo tanto, son aplicaciones, aquellos programas que nos permiten editar una imagen, enviar un correo electrónico, navegar en Internet, editar un documento de texto, chatear, etc.

Recordemos, que **un programa es el conjunto de instrucciones que ejecutadas en un ordenador realizarán una tarea o ayudarán al usuario a realizarla.**

2. Definición y estructura de programa y proceso

Nosotros, como programadores y programadoras, creamos un programa, escribiendo su código fuente; con ayuda de un compilador, obtenemos su código binario o interpretado. Este código binario o interpretado, lo guardamos en un fichero. Este fichero, es un fichero ejecutable, llamado comúnmente: ejecutable o binario.

Un ejecutable es un fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.

2. Definición y estructura de programa y proceso

De forma sencilla, se puede decir que **un proceso, es un programa en ejecución**. Pero, es más que eso, un proceso en el sistema operativo (SO), es una unidad de trabajo completa; y, el SO gestiona los distintos procesos que se encuentren en ejecución en el equipo. En siguientes apartados de esta unidad trataremos más en profundidad todo lo relacionado con los procesos y el SO. Lo más importante, es que diferenciemos que un ejecutable es un fichero y un proceso es una entidad activa, el contenido del ejecutable, ejecutándose.

Un proceso existe mientras que se esté ejecutando una aplicación. Es más, la ejecución de una aplicación, puede implicar que se arranquen varios procesos en nuestro equipo; y puede estar formada por varios ejecutables y librerías.

Al instalar una aplicación en el equipo, podremos ver que puede estar formada por varios ejecutables y librerías. Siempre que lancemos la ejecución de una aplicación, se creará, al menos, un proceso nuevo en nuestro sistema.

2. Definición y estructura de programa y proceso

Resumen programa, proceso y ejecutable

PROGRAMA: código + datos, almacenado en soporte digital que resuelve una necesidad concreta.

PROCESO: cuando un programa se ejecuta, es decir un programa en ejecución (código ejecutable + datos + pila del programa + contador de programa + puntero de pila + registros).

Los **procesos** son **entidades independientes** aunque ejecuten el mismo programa, es decir, podemos encontrarnos dos procesos que sean el mismo programa pero cada uno con su imagen de memoria, contador de programa o datos independientes.

EJECUTABLE: contiene la información necesaria para **CREAR** un proceso, es decir, ejecutable es el fichero que permite poner el programa en ejecución y convertirlo en proceso.

2. Definición y estructura de programa y proceso

Tipos de ejecutables según el código que contenga un ejecutable

- **Binarios.** Formados por un conjunto de instrucciones que **directamente son** ejecutadas por el procesador del ordenador. Este código se obtiene al compilar el código fuente de un programa y se guarda en un fichero ejecutable. Este código sólo se ejecutará correctamente en equipos cuya plataforma sea compatible con aquella para la que ha sido compilado (no es multiplataforma). Ejemplos son, ficheros que obtenemos al compilar un ejecutable de C o C++.
- **Interpretados.** Código que suele tratarse como un ejecutable, **pero no es código binario, sino otro tipo de código**, que en Java, por ejemplo se llama bytecode. Está formado por códigos de operación que tomará el intérprete (en el caso de Java, el intérprete es la máquina virtual Java o JRE). Ese intérprete será el encargado de traducirlos al lenguaje máquina que ejecutará el procesador. El código interpretado es más susceptible de ser multiplataforma o independiente de la máquina física en la que se haya compilado.

2. Definición y estructura de programa y proceso

Tipos de ejecutables según el código que contenga un ejecutable

- **Librerías.** Conjunto de funciones que permiten dar modularidad y reusabilidad a nuestros programas. Las hemos incluido en esta clasificación, porque su contenido es código ejecutable, aunque ese código sea ejecutado por todos los programas que invoquen las funciones que contienen. El conjunto de funciones que incorpora una librería suele ser altamente reutilizable y útil para los programadores; evitando que tengan que reescribir una y otra vez el código que realiza la misma tarea.

Ejemplo de librerías son: las librerías estándar de C, los paquetes compilados DLL en Windows; las API (Interfaz de Programación de Aplicaciones), como la J2EE de Java (Plataforma Java Enterprise Edition versión 2); las librerías que incorpora el framework de .NET; etc.

ÍNDICE

1. Repaso programación Java

- a) Utilizar los sangrados correctamente
- b) Comentar de manera correcta el código
- c) Nombrar las variables de forma estandarizada

2. Definición y estructura de programa y proceso

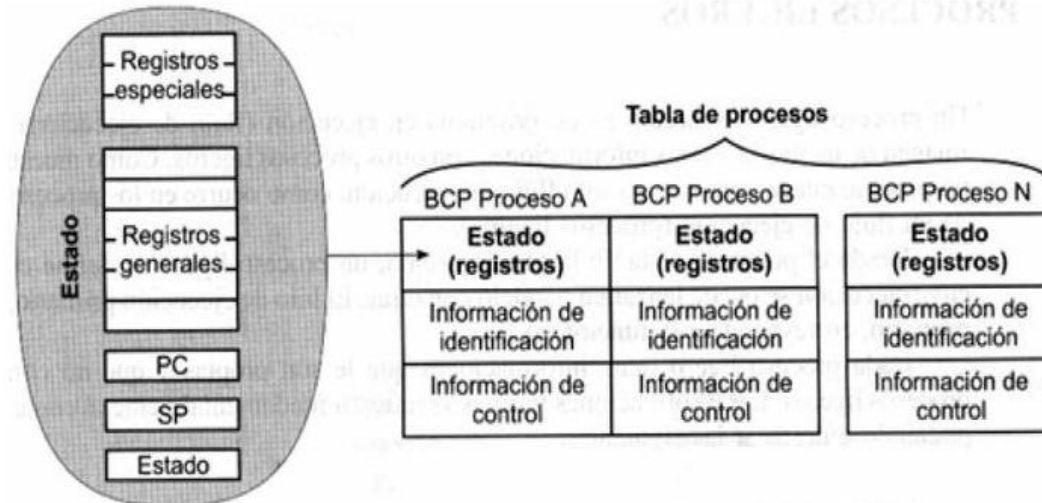
- a) Fichero ejecutable
- b) Proceso

3. Estados de un proceso en Linux

4. Diagrama de estados de un proceso

3. Estados de un proceso en Linux

Todos los programas se ejecutan y organizan como un conjunto de procesos y es el sistema operativo el que decide los estados a los que pasan cada proceso. Es el SO el encargado de decidir qué proceso puede entrar a ejecutarse o debe esperar. Cuando se suspende la ejecución de un proceso, luego deberá volver a arrancarse en el mismo estado en el que se encontraba antes de ser suspendido. Esto implica que debemos almacenar en algún sitio la información referente a ese proceso para poder luego restaurarla tal como estaba antes.



La **BCP (Bloque de Control de Proceso)** es donde se almacenará esa información: *identificador, estado, contador de programa (PC), registros de la CPU, puntero de pila (SP), prioridad del proceso, gestión de memoria, etc.*

3. Estados de un proceso en Linux

Herramientas gráficas para la gestión de procesos

Tanto los sistemas Windows como GNU/Linux proporcionan herramientas gráficas para la gestión de procesos. En el caso de Windows, se trata del Administrador de tareas, y en GNU/Linux del Monitor del sistema. Ambos, son bastante parecidos, nos ofrecen, al menos, las siguientes funcionalidades e información:

- Listado de todos los procesos que se encuentran activos en el sistema, mostrando su PID, usuario y ubicación de su fichero ejecutable.
- Posibilidad de finalizar procesos.
- Información sobre el uso de CPU, memoria principal y virtual, red.
- Posibilidad de cambiar la prioridad de ejecución de los procesos.

Abre el administrador de programas de tu ordenador. En MAX está en Herramientas del Sistema >> Monitor del sistema MATE. Observa qué procesos hay en estado de ejecución. Abre la calculadora, después revisa el administrador de programas, ¿se ha incluido en el listado? Finaliza la tarea del proceso que se ha abierto al abrir la calculadora.

3. Estados de un proceso en Linux

Un **proceso en Linux** genera un nuevo proceso para que realice una tarea determinada, y este nuevo proceso es considerado proceso “*hijo*” del proceso anterior, al que llamaremos “*padre*”. Esta **estructura de procesos padres e hijos forman un árbol jerárquico de procesos**, lo que nos da una idea de qué proceso generó a cuál otro.

En Linux los procesos pueden estar en diferentes estados. Mediante el comando *top* podemos ver su estado representado con una letra:

- **D Uninterruptible sleep**: generalmente el proceso se encuentra esperando una operación de E/S con algún dispositivo.
- **R Running**: el proceso se encuentra corriendo en el procesador.
- **S Interruptible sleep**: el proceso se encuentra esperando a que se cumpla algún evento, por ejemplo, que el planificador de procesos del kernel lo planifique para su ejecución.
- **T Stopped**, un proceso que ha sido detenido mediante el envío de alguna señal generalmente.
- **Z Defunct (“zombie”) process**, proceso terminado, pero cuyo padre aún sigue «vivo» y no ha capturado el estado de terminación del proceso hijo, y por consiguiente, no lo ha eliminado de la tabla de procesos del sistema.

ÍNDICE

- 1. Repaso programación Java
 - a) Utilizar los sangrados correctamente
 - b) Comentar de manera correcta el código
 - c) Nombrar las variables de forma estandarizada
- 2. Definición y estructura de programa y proceso
 - a) Fichero ejecutable
 - b) Proceso
- 3. Estados de un proceso en Linux
- 4. Diagrama de estados de un proceso**

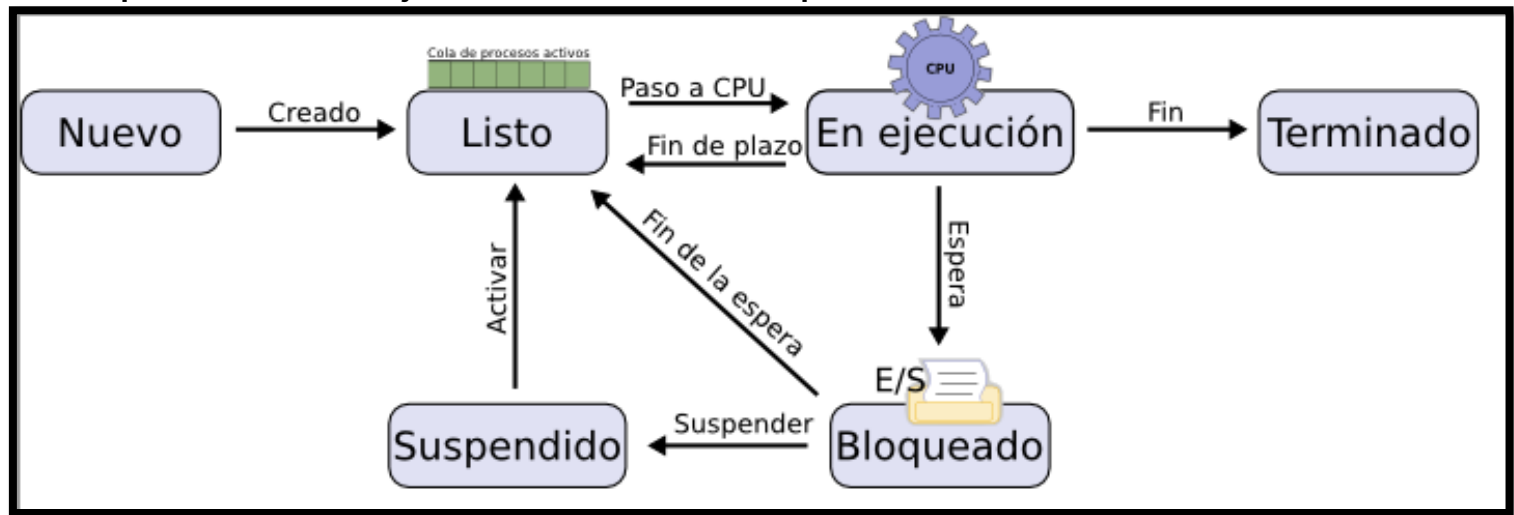
4. Diagrama de estados de un proceso

Estados en el ciclo de vida de un proceso

- **Nuevo.** Proceso nuevo, creado.
- **Listo.** Proceso que está esperando la CPU para ejecutar sus instrucciones.
- **En ejecución.** Proceso que actualmente, está en turno de ejecución en la CPU.
- **Bloqueado.** Proceso que está a la espera de que finalice una E/S.
- **Suspendido.** Proceso que se ha llevado a la memoria virtual para liberar, un poco, la RAM del sistema.
- **Terminado.** Proceso que ha finalizado y ya no necesitará más la CPU.

4. Diagrama de estados de un proceso

En el siguiente gráfico, podemos ver un esquema muy simple de cómo podemos planificar la ejecución de varios procesos en una CPU.



Transiciones que se producen entre uno u otro estado

1. Los procesos nuevos, entran en la cola de procesos activos en el sistema.
2. Los procesos van avanzando posiciones en la cola de procesos activos, hasta que les toca el turno para que el SO les conceda el uso de la CPU.
3. El SO concede el uso de la CPU, a cada proceso durante un tiempo determinado y equitativo, que llamaremos **quantum**. Un proceso que consume su *quantum*, es pausado y enviado al final de la cola.
4. Si un proceso finaliza, sale del sistema de gestión de procesos.

4. Diagrama de estados de un proceso

Otras situaciones y características de nuestros los procesos a considerar:

- Cuando un proceso, necesita datos de un archivo o una entrada de datos que deba suministrar el usuario; o, tiene que imprimir o grabar datos; cosa que llamamos 'el proceso está en una operación de entrada/salida' (E/S para abreviar). El proceso, queda bloqueado hasta que haya finalizado esa E/S. El proceso es bloqueado, porque, los dispositivos son mucho más lentos que la CPU, por lo que, mientras que uno de ellos está esperando una E/S, otros procesos pueden pasar a la CPU y ejecutar sus instrucciones.
- Cuando termina la E/S que tenga un proceso bloqueado, el SO, volverá a pasar al proceso a la cola de procesos activos, para que recoja los datos y continúe con su tarea (dentro de sus correspondientes turnos).

4. Diagrama de estados de un proceso

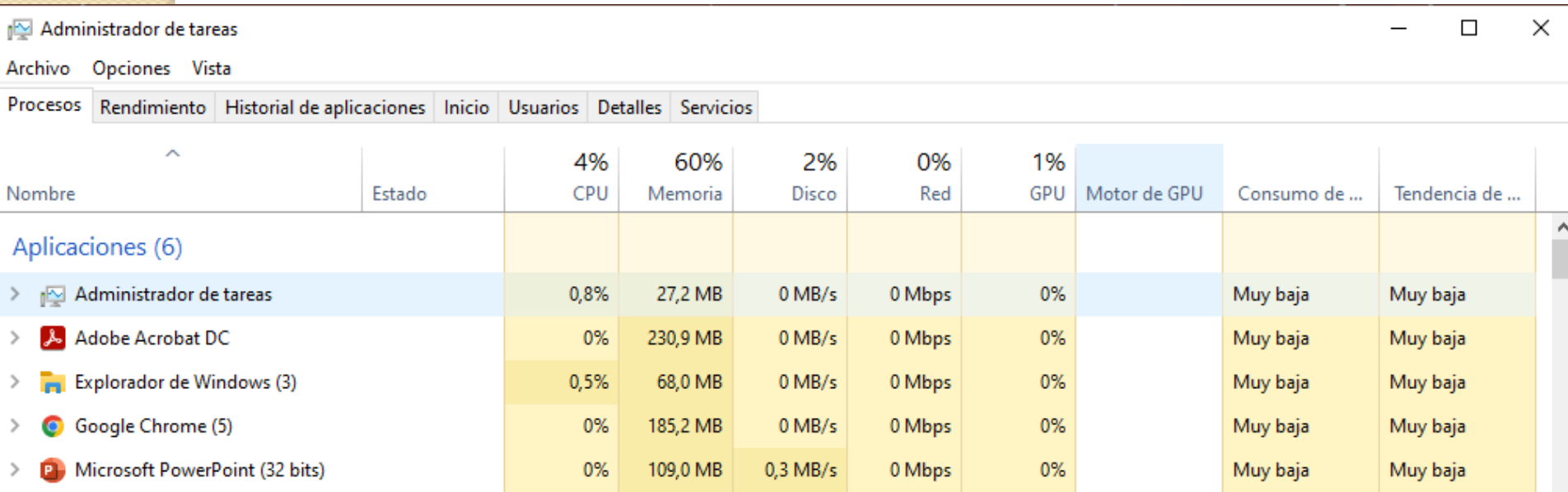
Otras situaciones y características de nuestros los procesos a considerar:

- Hay **procesos** en el equipo cuya **ejecución es crítica** para el sistema, por lo que, no siempre pueden estar esperando a que les llegue su turno de ejecución haciendo cola.
- Por ejemplo, el propio SO es un programa, se le da prioridad a los procesos del SO, frente a los procesos de usuario.

4. Diagrama de estados de un proceso

Procesos en los sistemas operativos. Windows

Para ver los procesos en Windows, iremos al *Administrador de Tareas*, pestaña *Procesos*.



Administrador de tareas									
Archivo Opciones Vista									
Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios									
Nombre	Estado	4% CPU	60% Memoria	2% Disco	0% Red	1% GPU	Motor de GPU	Consumo de ...	Tendencia de ...
Aplicaciones (6)									
> Administrador de tareas		0,8%	27,2 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
> Adobe Acrobat DC		0%	230,9 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
> Explorador de Windows (3)		0,5%	68,0 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
> Google Chrome (5)		0%	185,2 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
> Microsoft PowerPoint (32 bits)		0%	109,0 MB	0,3 MB/s	0 Mbps	0%		Muy baja	Muy baja

4. Diagrama de estados de un proceso

Procesos en los sistemas operativos. Windows

Para ver los procesos desde la línea de comandos, en CMD escribiremos ***tasklist***. Además al hacerlo por línea de comandos veremos el **PID**, el identificador del proceso.

 Símbolo del sistema

```
Microsoft Windows [Versión 10.0.18363.1256]
```

```
(c) 2019 Microsoft Corporation. Todos los derechos reservados.
```

```
C:\Users\ usuario >tasklist
```

Nombre de imagen	PID	Nombre de sesión	Núm. de ses	Uso de memor
=====	=====	=====	=====	=====
System Idle Process	0	Services	0	8 KB
System	4	Services	0	4.752 KB
Registry	96	Services	0	52.356 KB
smss.exe	504	Services	0	724 KB
csrss.exe	688	Services	0	2.884 KB
wininit.exe	784	Services	0	2.728 KB

4. Diagrama de estados de un proceso

Este SO es conocido por sus interfaces gráficas, el intérprete de comandos conocido como Símbolo del sistema, no ofrece muchos comandos para la gestión de procesos. Tendremos:

- **tasklist.** Lista los procesos presentes en el sistema. Mostrará el nombre del ejecutable; su correspondiente Identificador de proceso; y, el porcentaje de uso de memoria; entre otros datos.
- **taskkill.** Mata procesos. Con la opción /PID especificaremos el Identificador del proceso que queremos matar.

Ejercicio

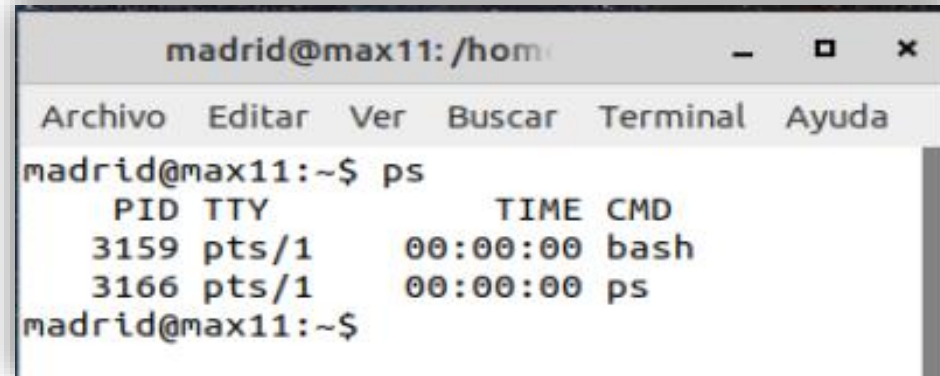
Abre la consola de comandos y lista los procesos. Después mata un proceso mediante su PID. Vuelve a listar los procesos y verifica que ya no está.

4. Diagrama de estados de un proceso

Procesos en los sistemas operativos. Ubuntu

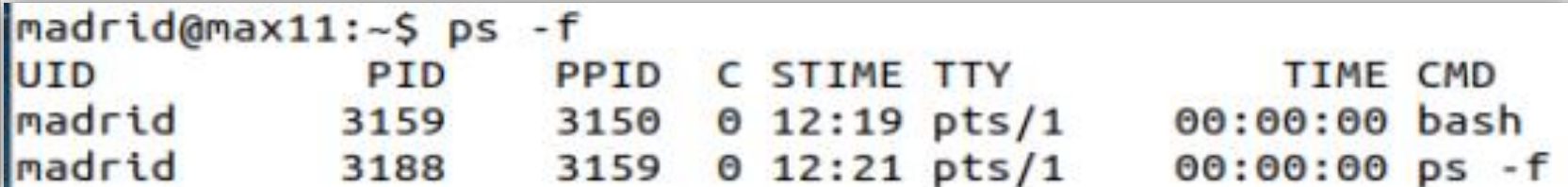
En Ubuntu podemos verlo en la terminal con el comando **ps**.

PID es el identificador del proceso. **TIME**: tiempo de ejecución asociado, es la cantidad total de tiempo CPU que el proceso ha utilizado desde que nació.



```
madrid@max11: /hom
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
madrid@max11:~$ ps
  PID TTY          TIME CMD
 3159 pts/1        00:00:00 bash
 3166 pts/1        00:00:00 ps
madrid@max11:~$
```

Si hacemos **ps -f** también aparecerá el **PPID**, el identificador del proceso padre. **C**: porcentaje de CPU utilizado por el proceso. **STIME**: inicio de ejecución. **UID**: nombre del usuario. **CMD**: nombre simple del proceso.



```
madrid@max11:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
madrid       3159    3150  0  12:19 pts/1        00:00:00 bash
madrid       3188    3159  0  12:21 pts/1        00:00:00 ps -f
```


4. Diagrama de estados de un proceso

Procesos en los sistemas operativos. Ubuntu

Con **ps -AF** aparecen todos los procesos lanzados en el sistema.

```
madrid@max11:~$ ps -AF
```

UID	PID	PPID	C	SZ	RSS	PSR	STIME	TTY	TIME	CMD
root	1	0	0	25553	9740	0	11:39	?	00:00:00	/sbin/init splash
root	2	0	0	0	0	0	11:39	?	00:00:00	[kthreadd]
root	3	2	0	0	0	0	11:39	?	00:00:00	[rcu_gp]
root	4	2	0	0	0	0	11:39	?	00:00:00	[rcu_par_gp]
root	6	2	0	0	0	0	11:39	?	00:00:00	[kworker/0:0H-kblockd]
root	9	2	0	0	0	0	11:39	?	00:00:00	[mm_percpu_wq]
root	10	2	0	0	0	0	11:39	?	00:00:00	[ksoftirqd/0]
root	11	2	0	0	0	0	11:39	?	00:00:00	[rcu_sched]
root	12	2	0	0	0	0	11:39	?	00:00:00	[migration/0]
root	13	2	0	0	0	0	11:39	?	00:00:00	[idle_inject/0]

Ejercicio

Abre la consola de comandos y lista los procesos. Abre la calculadora. Después mata el proceso de la calculadora mediante su PID con el comando kill. Vuelve a listar los procesos y verifica que ya no está.

4. Diagrama de estados de un proceso

Cambio de contexto

De forma general, el ***contexto de un proceso*** en un cierto instante de tiempo se puede definir como ***la información relativa al proceso que el núcleo debe conocer para poder iniciar o continuar su ejecución***. Cuando se ejecuta un proceso se dice que el sistema se ejecuta en el contexto de dicho proceso. Por lo que cuando el núcleo decide pasar a ejecutar otro proceso debe de ***cambiar de contexto***, de forma que el sistema pasará a ejecutarse en el contexto del nuevo proceso.

El contexto de un proceso se puede considerar como la unión del contexto a nivel de usuario, contexto de registros y contexto a nivel del sistema.

4. Diagrama de estados de un proceso

Cambio de contexto

Dentro de la información que se guarda, destaca:

- El **Contador de Programa**, en cada instante almacena la dirección de la siguiente instrucción a ejecutar.
- El **Puntero a Pila**, en cada instante apunta a la parte superior de la pila del proceso en ejecución. En la pila de cada proceso es donde será almacenado el contexto de la CPU. Y de donde se recuperará cuando ese proceso vuelva a ejecutarse.

El cambio de contexto que se hace al cambiar un proceso es tiempo perdido, ya no se hace trabajo útil. Cambiar el estado del proceso, el estado del procesador (cambio valores de registro) e información de la gestión de memoria, por muy rápido que se haga si se hace con mucha frecuencia puede provocar una **ralentización del sistema**, por eso **tener muchos programas abiertos** provoca una **disminución** importante en el **rendimiento del sistema**.