# Gestión del procesos



## 1. Concepto de proceso

Para el almacenamiento permanente de los programas se utiliza la memoria externa. Un programa permanecerá ahí cuando el ordenador esté apagado o, bien cuando esté encendido, mientras no sea ejecutado. Todos los recursos que consumirá será el espacio en disco que ocupe.

**Cuando un programa deba ser ejecutado, tanto sus instrucciones como sus datos serán cargadas en memoria principal** por el sistema operativo. Pues bien, desde el momento en que el programa se carga en memoria se convierte en un proceso. Por lo tanto, un **proceso es un programa en ejecución**.

A diferencia de un programa, que el único recurso que utiliza es la parte del espacio de almacenamiento de disco que ocupa, un **proceso necesita muchos más recursos** como la memoria principal, el procesador y sus registros. Un **programa es una entidad estática**, que no cambia, mientras que un **proceso es una entidad dinámica**, en el sentido de que, a lo largo de su ejecución, va atravesando por diferentes **estados**.

Los sistemas operativos **multitarea** (o **multiprogramados**), que se caracterizan por la capacidad de trabajar con **varios procesos simultáneamente** (**concurrencia**), realizan una serie de funciones derivadas del funcionamiento de dichos procesos:

- ✔ Establecer las **prioridades** de ejecución de los procesos.
- ✔ **Asignar los recursos** a los procesos. Esta cuestión, tal como veremos, es de gran relevancia, especialmente cuando se trata de recursos escasos como puede ser el procesador. Los procesos **compiten** por los recursos y es el sistema operativo el responsable de asignarlos de forma ordenada.

- ✔ **Impedir que un proceso interfiera** con otros procesos o invada sus recursos. Por ejemplo, que no acceda a áreas de memoria principal de otros procesos.

- ✔ Proporcionar mecanismos que permitan la **comunicación y colaboración entre procesos**.

Puesto que el sistema operativo es un conjunto de programas, cuando éstos se ejecutan también se convierten en procesos. Sin embargo, existen diferencias entre entre los procesos de usuario y los del sistema operativo:

- ✔ Los primeros procesos que existen cuando, al arrancar el ordenador, el sistema operativo toma el control, son procesos del **propio sistema operativo**.

- ✔ El procesador puede funcionar en dos modos:

  - **Modo kernel o modo privilegiado**. En este modo los procesos tienen libertad para acceder a todos los recursos hardware del sistema. Es el modo en el que se ejecutan los **procesos del sistema operativo**.

  - **Modo usuario**. Este modo, en el que se ejecutan los **procesos de usuario**, impone restricciones de acceso a los recursos hardware.

Para gestionar los procesos, el sistema operativo mantiene para cada proceso una **estructura de datos** denominada **bloque de control de proceso** (**PCB**), que permite guardar diferente información relativa al proceso, como un **identificador del mismo, recursos asignados, prioridad, estado y otra información** que veremos más abajo. El conjunto de PCBs se almacena en una tabla denominada **tabla de procesos**.

# 2. Gestión de procesos

## Bloque de Control de Proceso (BCP)

Cada vez que se ejecuta un programa, además de ser ubicadas en memoria las instrucciones que lo componen y sus datos, a dicho proceso el sistema operativo le asocia una estructura de datos. A esta estructura se le denomina **Bloque de Control de Proceso** (**BCP-PCB**). Este bloque contiene la siguiente información:

- Estado actual del proceso
  - Preparado
  - En ejecución
  - Bloqueado
- Identificación del proceso.
- Prioridad del proceso
- Ubicación en memoria
- Recursos utilizados

El funcionamiento de un sistema operativo **monotarea** (o **monoprogramado**) es fácil de comprender sin necesidad de recurrir al concepto de proceso: un programa se carga en memoria para ser ejecutado y permanece en el sistema hasta que su ejecución concluya. Cuando acabe se reemplaza por otro sin mayor problema.

En cambio, en un sistema **multiprogramado** se cargan en memoria varios programas de tal forma que el uso de la CPU se conmuta continuamente de un proceso a otro durante intervalos de tiempo relativamente pequeños, dando la sensación de que se están ejecutando en paralelo, cuando realmente no es así.

La tarea de quitar la CPU a un proceso para asignarla a otro se denomina **cambio de contexto**. Esta operación implica **dos acciones por parte del sistema operativo**:

1. Guardar en el **PCB** del proceso al que se le retira el uso de la CPU los valores que tienen los elementos del sistema utilizados por dicho proceso (por ejemplo, el **contador de programa y otros registros de la CPU**, etc.) en ese instante.

2. **La restauración del estado de los recursos del sistema** para el proceso entrante, desde su PCB, a fin de que pueda continuar su ejecución a partir del punto en que fue interrumpida.

## Estados de un proceso

Los procesos se pueden encontrar en alguno de los siguientes estados:

- ✔ **En ejecución**. El proceso está utilizando la CPU. Si el sistema únicamente tiene una CPU de un solo núcleo, en cada instante sólo puede haber un proceso en este estado.

- ✔ **Bloqueado**. Desde un punto de vista lógico la ejecución del proceso no puede continuar. Esta circunstancia ocurre habitualmente porque está realizando una operación de entrada/salida.

- ✔ **Preparado**. El proceso puede continuar con su ejecución, pero la CPU está siendo utilizada por otro.

Las transiciones entre los distintos estados se describen a continuación:

1. Tiene lugar en el momento en que el proceso que dispone de la CPU no puede continuar su ejecución, normalmente por tener que efectuar E/S.

2. El sistema operativo considera que el proceso en ejecución ya ha dispuesto del procesador durante suficiente tiempo, y decide que es el momento de asignarlo a otro.

3. El proceso vuelve a recibir tiempo de uso del procesador después de que el resto de procesos haya consumido el que tenían asignado.

4. Ha concluido la operación por la que el proceso estaba bloqueado, de forma que su ejecución vuelve a ser posible. Si cuando se produzca esta transición no hay ningún proceso en ejecución, inmediatamente se producirá la transición 3.


## Carga o ejecución del proceso y expulsión y finalización del mismo

Cuando hacemos click en un **programa ejecutable** (introducción de comando en shell de UNIX o en MS-DOS en command.com o en una interfaz gráfica en un fichero .exe), se carga un proceso del propio sistema operativo (denominado **cargador**) que realiza las siguientes tareas:

1. Crea el **Bloque de Control de Proceso** (BCP).

- Le asigna el identificador al proceso (programa) que se va a ejecutar (PID).
- Le asigna una prioridad base (la establecida por el sistema operativo).
- Le asigna todos los recursos a excepción de la CPU.

2. Se inserta en la **tabla de procesos del sistema**.

3. Se carga en **memoria virtual** (depende de la técnica de gestión de memoria).

4. Cuando tiene todos los recursos asignados (excepto la CPU), se pone el **campo de estado del BCP en preparado y se le ingresa en la cola de procesos listos** para hacer uso de la CPU. El proceso del sistema que controla la cola de la CPU se denomina **planificador** y es el encargado de elegir qué proceso será el siguiente en ser atendido por la CPU según las prioridades marcadas.

5. Una vez cargado y lanzado **el planificador continua con la planificación** de su cola de procesos.

Como dijimos, debemos compartir el tiempo de CPU entre los procesos. El tiempo compartido consiste en **dividir el tiempo de ejecución del procesador en minúsculos intervalos de tiempo** (**quantums**) e ir asignando cada uno de estos intervalos de ejecución a cada proceso que está en ejecución.

El sistema operativo le asigna un **código identificador de proceso** (**un número único**) a cada uno de los procesos (denominado **PID**). Este código es un número que el sistema operativo le asigna según unas **prioridades y parámetros** del propio sistema operativo y lo distingue del resto de procesos.

- ✔ En sistemas Windows podremos ver el PID de los procesos mediante la orden **taskmgr** o **Ctrl + Alt + Esc** y en **Opciones** → seleccionar columnas **PID**.

- ✔ En sistemas Unix y Linux mediante la orden **PS**.

### Planificador (*scheduler*) y despachador (*dispatcher*)

El **planificador** es el encargado de determinar **qué proceso pasará al estado de activo de entre todos los procesos que están en el estado de preparado**. La elección de los algoritmos de planificación a utilizar por el planificador se realiza teniendo en cuenta sus características frente a los criterios de diseño elegidos.

El **despachador** o **activador**, se encarga de **poner en ejecución el proceso que selecciona el planificador** de entre los preparados para ejecutarse. Esto consiste en tomar todos los datos necesarios para la ejecución de ese proceso y ponerlos a disposición del procesador.

## 3. Gestión de la CPU

En un sistema multiprogramado es normal que hayan varios procesos, cuya ejecución pueda continuar, que se encuentren a la espera de recibir la CPU (en estado **preparado**). El **planificador** es la parte del sistema operativo encargada de decidir cuál de esos procesos es el siguiente en reanudar su ejecución. El conjunto de criterios que éste utiliza para tomar tal decisión se denomina **algoritmo de planificación**.

Los criterios deseables en cualquier algoritmo de planificación son los siguientes:

- ✔ **Equidad**: que se asigne el procesador a cada proceso de forma equitativa.

- ✔ **Eficiencia**: que se mantenga ocupado el procesador el 100% del tiempo.

- ✔ **Tiempo de respuesta**: que se minimice el tiempo de respuesta para usuarios interactivos.

- ✔ **Tiempo de procesamiento global**: que se minimice el tiempo que han de esperar los usuarios de trabajos por lotes para obtener los resultados.

- ✔ **Rendimiento**: que se maximice el número de trabajos procesados en cada unidad de tiempo.

Es fácil observar que algunos de los objetivos son contrapuestos. Por ejemplo, para minimizar el tiempo de respuesta para usuarios interactivos, el planificador no deberá realizar ningún procesamiento por lotes.

Los ordenadores tienen un temporizador hardware, o **reloj**, que produce interrupciones periódicas. Éste suele tener una frecuencia de **50 o 60 Hz**. Con cada **interrupción de reloj**, **el sistema operativo recupera el control** y decide si debe dejar que el proceso actual continúe ejecutándose, o si, por el contrario, éste ya ha utilizado el procesador durante un tiempo suficientemente largo y, por tanto, es momento de suspender su ejecución y ceder el uso del mismo a otro proceso.

# 4. Algoritmos de planificación del procesador

## FCFS (*First Come First Served*)

El primero de los algoritmos es el **FCFS** (***First Come First Served***). Este algoritmo se comporta de una manera muy simple. Los procesos se van a ordenar por orden de llegada a la cola de activo (**Cola FIFO - *First In First Out***).

Un ejemplo para comprenderlo mejor: imagina que vas a ir al cine con tus padres y tienes que comprar tres entradas en un cine con una única taquilla. La forma que habitualmente tenemos de comportarnos es que cuando alguien llega a la cola se pone el último y espera su turno. Una vez que eres el primero de la cola y no hay nadie en la taquilla pasas a comprar las entradas y puedes comprar tantas como desees ya que no existe ninguna restricción en cuanto el número de entradas a comprar. Cuando terminas de comprar las entradas abandonas la taquilla para que el primero de la cola pase a comprar.

Si trasladamos este ejemplo a los procesos vamos a pensar que tenemos **5 procesos** (**A, B, C, D y E**) que están interesados en comprar entradas. Para poder saber en qué momento llegan a la cola del cine tenemos lo que se denomina **ciclo de llegada**, que medimos en **unidades de tiempo o ciclos de reloj**. Por otra parte tenemos los **ciclos totales de CPU**, que no es más que el número de ciclos de reloj o unidades de tiempo que necesita un proceso para ejecutarse de forma normal (si piensas en el ejemplo sería el número de entradas que necesitas - recuerda que la taquillera nos imprimirá una entrada en cada ciclo de reloj).

## SJF (*Shortest Job First*)

El segundo de los algoritmos es el **SJF** (***Shortest Job First***). Este algoritmo se comporta de una manera muy simple. Los **procesos de la cola** se ordenan por el **número total de ciclos de reloj** que necesitan.

Si volvemos al ejemplo del cine quiere decir que cada vez que alguien llega a la cola, **la cola se reordenará** por número total de ciclos de reloj o unidades de tiempo (duración). Aquellos que necesiten menos entradas estarán al principio de la cola y aquellos que necesiten más estarán al final. Si hay alguna persona que necesite el mismo número de entradas que otra entonces tendremos en cuenta que irá primero de los dos el que llegó primero a la cola.

## RR (*Round Robin*)

El tercero de los algoritmos es el **RR** (***Round Robin***). Este algoritmo se comporta de una forma un poco más compleja. Este algoritmo trata de dar ventaja a aquellos procesos que necesiten menos tiempo (menos

*quantums*) para ejecutarse pero dando a todos la posibilidad de entrar durante un ***quantum*** (uno o más ciclos de reloj o unidades de tiempo) de forma circular.

Imaginemos un concierto de un artista muy famoso o un partido de fútbol muy importante y en la taquilla ves un cartel que dice “dos entradas por persona”. Este algoritmo trabaja así. Imagina que vas a ir con tus padres y necesitas tres entradas. Esperarás en la cola hasta que te toque ir a la taquilla y comprarás dos entrada. Acto seguido pasarás de nuevo al final de la cola y comprarás la tercera cuando te toque.

Para simplificar el algoritmo vamos a seguir dos sencillas preguntas y siempre en el mismo orden.

**¿Hay alguien esperando en la cola de espera?**

- **Si hay alguien esperando** haremos lo siguiente:
  - El que esté ejecutándose que pase al final de la cola.
  - El primero de la cola pasará a ejecutarse.

- **Si no hay nadie esperando**:
  - El proceso que está ejecutándose puede quedarse un **quantum** más.

**¿Hay alguien que quiera entrar en la cola de espera?**

- **Si hay algún proceso nuevo** (que quiera formar parte de la cola) se insertará el último de la cola.

## SRTF (*Shortest Remaining Time First*)

El cuarto y último de los algoritmos es el **SRTF** (***Shortest Remaining Time First***). Este algoritmo se comporta de una manera muy similar al SJF pero en este caso se tiene en cuenta el número de ciclos de reloj que le restan para finalizar de forma normal incluyendo al proceso que está ejecutándose en ese momento.

Si volvemos al ejemplo del cine quiere decir que cada vez que alguien llega a la cola, **la cola se reordenará** por el número de entradas que les restan por comprar. Aquellos que necesiten menos entradas estarán al principio de la cola y aquellos que necesiten más estarán al final pero teniendo en cuenta también al que está comprando en la taquilla (**apropiativo**). Si hay alguna persona que necesite el mismo número de entradas que otra, entonces tendremos en cuenta que irá primero el que llegó primero a la cola.