## ****PROCESOS****

Un **proceso** es una instancia de un programa en ejecución. Incluye el código que se ejecuta, registros, contador de programa, áreas de memoria y todo lo necesario para caracterizar y gestionar dicha ejecución.

Varios procesos pueden compartir un mismo procesador, por lo que se utiliza algún **algoritmo de planificación (o scheduler)** para decidir cuál proceso se ejecuta y cuándo finaliza o se interrumpe.

**Ejemplos de algoritmos de planificación:**

* **FCFS** (First Come, First Served)
* **Round Robin**
* **Prioridades**
* **SJF** (Shortest Job First)

### **MODOS DE EJECUCIÓN**

* **Modo usuario**: permite la ejecución de instrucciones que no afectan a otros procesos ni al núcleo del sistema operativo.
* **Modo kernel**: permite la ejecución de todas las instrucciones, incluidas las privilegiadas (acceso a hardware, gestión de memoria, etc.).

### **CREACIÓN DE PROCESOS**

1. **Inicialización del sistema**
2. **Llamadas al sistema** para crear un proceso (ej.: fork + exec, createProcess)
3. **Petición del usuario**: al lanzar una nueva aplicación desde la interfaz gráfica o consola.
4. **Inicio por lotes**: en servidores, mediante un sistema de **cola de trabajos** (batch processing).

### **FINALIZACIÓN DE PROCESOS**

**Voluntarias:**

1. **Terminación normal** (ej.: exit, exitProcess)
2. **Terminación por error**: por ejemplo, falta un parámetro necesario.

**Involuntarias:**

3. **Error fatal**: excepciones, violaciones de segmento (segmentation fault), etc.  
4. **Terminado por otro proceso** (ej.: kill, terminateProcess)

## ****BLOQUE DE CONTROL DEL PROCESO****

Se lo conoce como **BCP** (en español) o **PCB** (Process Control Block, en inglés). Es un **registro especial** donde el sistema operativo agrupa toda la información necesaria sobre un proceso en particular.

Cada vez que se **crea un proceso**, el sistema operativo genera su correspondiente BCP. Este actúa como **descripción en tiempo de ejecución** durante toda la vida útil del proceso.  
Cuando el proceso finaliza, el BCP se elimina y el registro puede ser reutilizado por otro proceso.

### **Información que almacena el BCP**

1. **ID del proceso**
2. **Estado del proceso** (Ej.: listo, bloqueado, ejecutando, etc.)
3. **Contador de programa** (dirección de la próxima instrucción a ejecutar)
4. **Registros de CPU** (se usan en el cambio de contexto)
5. **Espacio de direcciones de memoria**
6. **Prioridad del proceso**, si se utiliza planificación por prioridades
7. **Lista de recursos asignados** (archivos, dispositivos, etc.)
8. **Estadísticas del proceso** (tiempo de CPU, tiempos de espera, etc.)
9. **Datos del usuario** (si aplica)
10. **Permisos asignados**
11. **Señales pendientes** (signals que aún no fueron servidas)

## ****CAMBIO DE CONTEXTO**** (Context Switch)

Es la ejecución de una **rutina del kernel** que guarda el estado del proceso en ejecución y lo reemplaza por otro proceso, permitiendo así la multitarea.

### **Motivación del cambio de contexto**

En sistemas con un solo procesador, **solo puede ejecutarse un proceso a la vez**.  
Para simular la ejecución simultánea de varios procesos (**concurrencia**), el sistema detiene un proceso y reanuda otro, **sin perder su estado**.  
Este intercambio continuo entre procesos da la ilusión de ejecución paralela.

### **Funcionamiento del cambio de contexto**

Requiere intervención del **hardware**, normalmente a través de **interrupciones**.  
Pasos:

1. **Guardar el estado** del proceso actual (BCP)
2. **Seleccionar** otro proceso listo para ejecutarse
3. **Restaurar el estado** del proceso seleccionado
4. **Reanudar la ejecución** del nuevo proceso

## ****ESTADOS DE UN PROCESO****

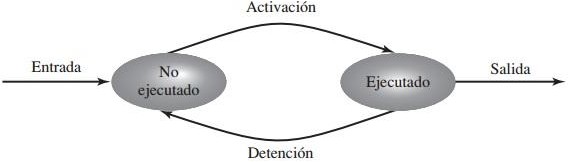
Un proceso puede encontrarse en distintos **estados** a lo largo de su vida, dependiendo de su situación dentro del sistema operativo.

Cuando ocurre un **evento** (como una interrupción, finalización de E/S, etc.), el proceso puede **cambiar de estado**, lo que se denomina **transición de estado**.

Ejemplos de estados:

* **Nuevo**
* **Listo**
* **En ejecución**
* **Bloqueado**
* **Finalizado**

**MODELO DE 2 ESTADOS**



Este modelo simplificado considera solo **dos estados posibles** para un proceso:

### **No ejecutado (o inactivo)**

El proceso está **esperando** para ser ejecutado.  
Aquí se encuentran todos los procesos que han sido **creados**, pero **aún no están siendo ejecutados**.

### **Ejecutado (o activo)**

El proceso **está siendo ejecutado** en la CPU.

## ****MODELO DE 5 ESTADOS****

En sistemas operativos modernos, un proceso puede transitar por **cinco estados principales**, según su situación dentro del sistema y la interacción con la CPU, memoria y dispositivos.

### **Nuevo**

El proceso **acaba de ser creado**, pero aún **no ha sido admitido** en el grupo de procesos ejecutables.  
Aunque su **BCP ya fue generado**, todavía **no está cargado en memoria principal**.

### **Listo**

El proceso **está preparado para ejecutarse** tan pronto como el sistema operativo le asigne el procesador.  
Puede haber **muchos procesos en este estado al mismo tiempo**.

### **Ejecutando**

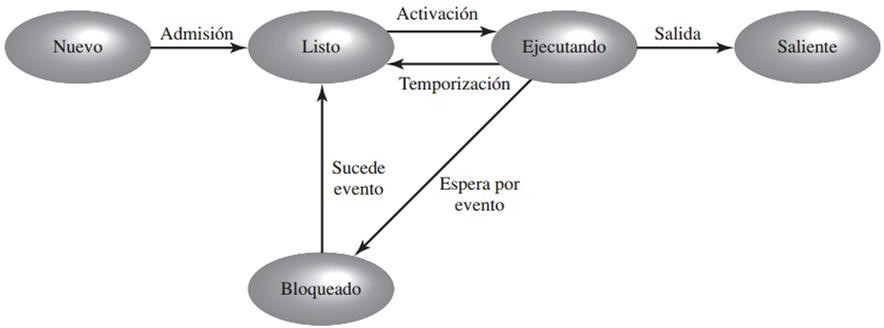
El proceso **está usando activamente el procesador**.  
Su código se ejecuta y realiza operaciones. En un sistema monoprocesador, **solo puede haber un proceso en este estado a la vez**.

### **Bloqueado**

El proceso **no puede continuar ejecutándose** hasta que ocurra un evento específico, como **finalizar una operación de entrada/salida**.

### **Saliente**

El proceso ha **finalizado su ejecución**, ya sea de forma **normal** o por **aborto/error**.  
En este estado el sistema operativo **libera sus recursos** y elimina su BCP.



## ****MODELO DE 7 ESTADOS****

Este modelo amplía el de 5 estados incluyendo **estados de suspensión**, permitiendo que procesos que no caben en la memoria principal sean **almacenados en disco temporalmente**.

### **Nuevo**

Proceso recién creado, con su BCP generado, pero aún no está en memoria principal.

### **Listo**

Proceso cargado en memoria, **espera que se le asigne la CPU**.

### **Listo/Suspendido**

Proceso que estaba listo, pero fue **suspendido y movido a disco**.

### **Ejecutando**

Proceso que está usando el procesador actualmente.

### **Bloqueado**

Proceso **esperando un evento** (por ejemplo, fin de operación E/S).

### **Bloqueado/Suspendido**

Proceso que estaba bloqueado y fue **movido a disco**.

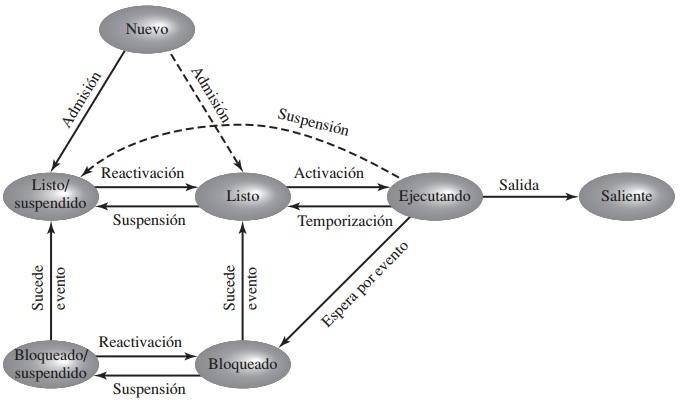
### **Saliente**

Proceso que ha **finalizado o fue terminado**, y está liberando recursos.

### **PROCESOS SUSPENDIDOS**

Los **procesos suspendidos** tienen estas características clave:

1. **No están en memoria principal**, por lo que **no pueden ejecutarse inmediatamente**.
2. Pueden estar **bloqueados o listos**, pero la **condición de suspensión es independiente** del evento que espera.
3. Fueron suspendidos por un **agente externo**: el propio proceso, su padre o el sistema operativo.
4. Solo pueden reactivarse cuando **ese agente explícitamente lo indique**.



**PLANIFICACIÓN**

* El **planificador (scheduler)** del sistema operativo elige qué algoritmo de planificación usar.
* Los **algoritmos de planificación** se aplican cuando hay **dos o más procesos** en estado **Listo**.

### **COLAS DE PLANIFICACIÓN**

El sistema operativo organiza los procesos mediante **colas**:

1. **Cola de trabajos**:  
   Procesos en almacenamiento secundario esperando entrar a memoria principal.
2. **Cola de procesos listos**:  
   Procesos cargados en memoria, esperando su turno para ejecutarse.
3. **Cola de dispositivos**:  
   Cada dispositivo tiene su propia cola de procesos esperando por él.

### **FUNCIONES DEL PLANIFICADOR DE PROCESOS**

1. **Llevar** control del estado de cada proceso.
2. **Decidir** qué proceso usará el procesador y por cuánto tiempo, con base en un criterio.
3. **Asignar** el procesador al proceso seleccionado.
4. **Quitar** el procesador al proceso actual cuando termine su turno o se bloquee.

### **TIPOS DE PLANIFICACIÓN (por plazo)**

1. **Planificación a largo plazo**:
   * Ocurre cuando se **crea un proceso**.
   * Se deciden criterios como **la prioridad** o el **quantum** (tiempo máximo de uso del procesador).
2. **Planificación a corto plazo**:
   * Se aplica **cada vez que un proceso abandona la CPU**.
   * Se decide **qué proceso la ocupará**, según la política de planificación y los parámetros definidos.
3. **Planificación a mediano plazo**:
   * Intervienen otras partes del sistema operativo de forma **indirecta** (como la técnica de swap).
   * Al sacar un proceso de la memoria, este **ya no es planificable**, lo que afecta la planificación activa.

### **TIPOS DE PLANIFICACIÓN (por control)**

**No Apropiativo**

* El proceso **mantiene el uso de la CPU** hasta que **finaliza voluntariamente**.
* El sistema operativo **no puede interrumpirlo**.
* *Ejemplos:*
  + FCFS (First Come First Served)
  + SJF (Shortest Job First)
  + Prioridades (si no se permite expulsión)

**Apropiativo**

* El sistema operativo **puede interrumpir** al proceso en ejecución.
* Se transfiere la CPU a otro proceso según prioridad o tiempo.
* *Ejemplos:*
  + Round Robin
  + SRTF (Shortest Remaining Time First)
  + Prioridades (si se permite expulsión)

### **EVALUACIÓN DEL RENDIMIENTO DE LA PLANIFICACIÓN**

* **Equidad**:  
   Asegura que todos los procesos tengan acceso a la CPU de forma justa.
* **Eficiencia**:  
   Busca una **utilización máxima del CPU (idealmente 100%)**.
* **Tiempo de retorno/finalización/ejecución**:  
   Tiempo total que tarda un proceso en completarse.
* **Tiempo de respuesta**:  
   Tiempo entre la solicitud del usuario y la primera respuesta del sistema (clave en sistemas \_interactivos).
* **Tiempo de espera**:  
   **Tiempo que un proceso pasa esperando en la cola de listos.**
* **Rendimiento (productividad)**:  
   **Cantidad de procesos completados por unidad de tiempo.**

**ALFORITMO FCFS - (First Come, First Served)**

Significado: FCFS – First come, first server

**Tipo**: No apropiativo.

Ejecución: El primer proceso que entró en la cola de procesos listos es el

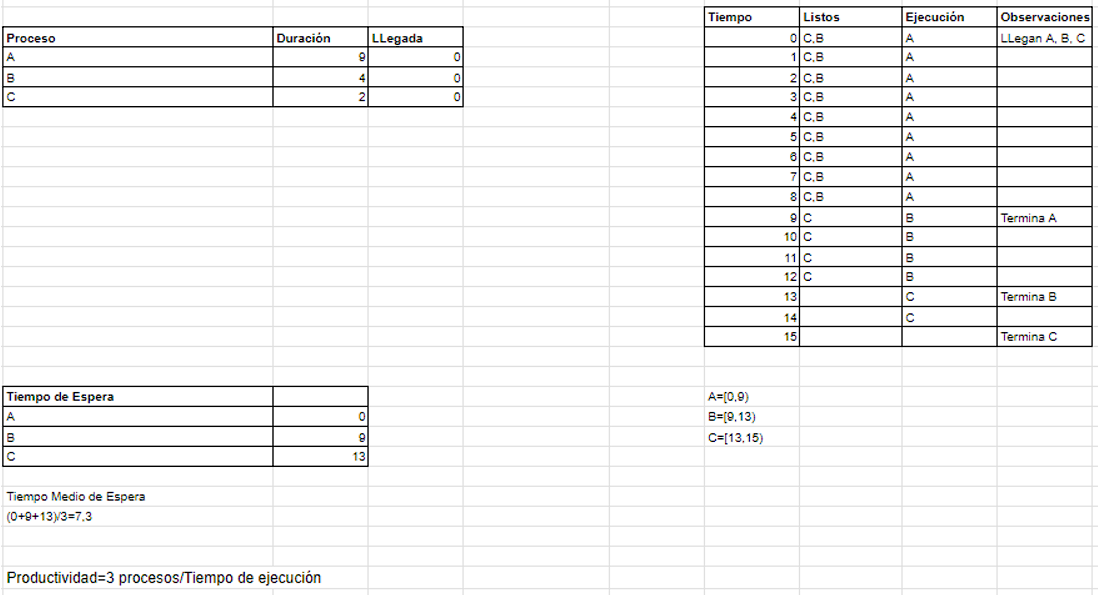
Implementación: Utiliza una **cola FIFO (First In, First Out)**.

Modo de ejecución:

* Es un algoritmo de **ejecución hasta terminación** (no se interrumpe hasta que el proceso finaliza).

Observación importante:

* **Sensible al orden de llegada** de los procesos: si un proceso largo llega primero, puede generar mayor tiempo de espera para los siguientes (efecto *convoy*).



### **Algoritmo SJF (Shortest Job First)**

**Significado**: Short Job First - "Primero el trabajo más corto".

**Tipo**: No apropiativo (en su versión básica).

**Funcionamiento**:

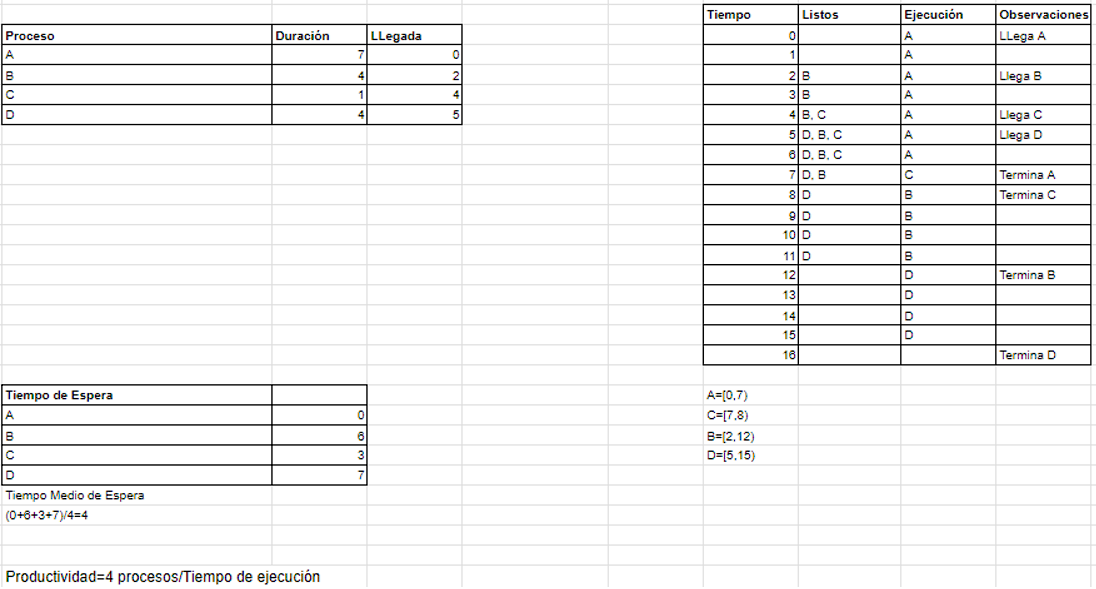
* + A cada proceso se le **asocia la duración estimada de su próxima ráfaga de CPU**.
  + Se le asigna la CPU al proceso con **menor duración de ráfaga**.
  + Si hay dos procesos con la misma duración, se usa **FCFS** como desempate.

**Requisito**:

* + Se necesita **conocer de antemano la duración** de cada proceso, lo cual **no siempre es posible**.

**Problema**:

* + Riesgo de **inanición (starvation)**: si llegan muchos trabajos cortos, los largos **pueden quedar esperando indefinidamente**.



### **Algoritmo SRTF**

**Significado**: (Shortest Remaining Time First) - "Primero el menor tiempo restante".

**Tipo**: Apropiativo.

**Funcionamiento**:

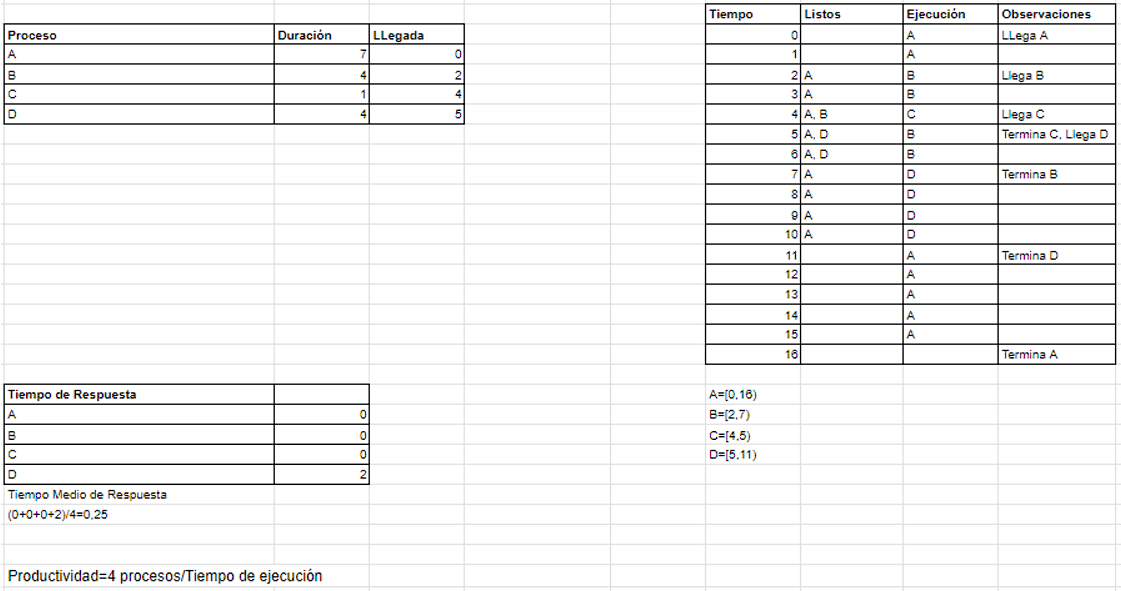
* + El planificador **elige el proceso con menor tiempo de ejecución restante esperado**.
  + Si llega un nuevo proceso con una ráfaga menor, **puede interrumpir al proceso actual**.

**Requisito**:

* + El sistema debe contar con una **estimación de la duración de los procesos**, lo que no siempre es exacto o fácil de obtener.

**Problema**:

* + Existe riesgo de **inanición (starvation)** para los procesos largos si constantemente llegan procesos más cortos.



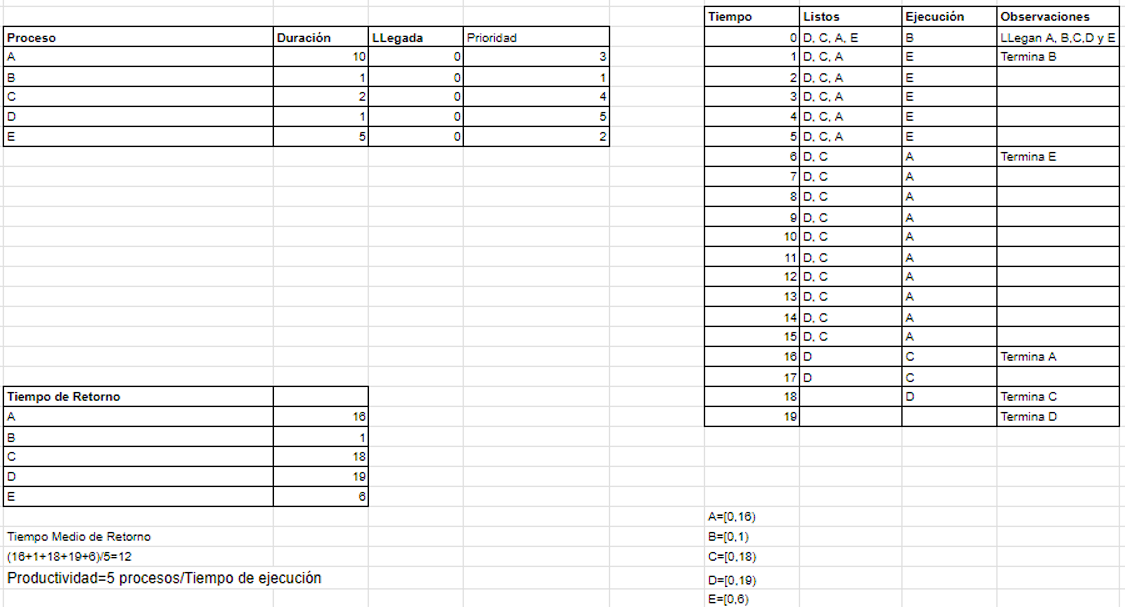
### **ALGORITMO - PRIORIDADES**

**Funcionamiento**:

* + A cada proceso se le **asigna una prioridad**.
  + El proceso con **mayor prioridad** (es decir, con **número más bajo**) accede primero a la **CPU**.
  + Si hay dos procesos con la **misma prioridad**, se utiliza **FCFS** como desempate.

**Observaciones**:

* + Puede ser **apropiativo o no apropiativo**, dependiendo de si se permite expulsar a un proceso por otro de mayor prioridad.
  + Este algoritmo puede causar **inanición (starvation)** si procesos con baja prioridad nunca reciben CPU.



### **Algoritmo Round Robin (RR)**

También llamado **Turno Rotatorio** o **Circular**.

Ideal para **sistemas de tiempo compartido**, donde múltiples usuarios deben obtener respuesta rápida.

Es una variante de **FCFS**, pero con **expropiación** (interrupción).

### **Funcionamiento**:

* A cada proceso se le asigna un **quantum**: tiempo máximo que puede usar la CPU.
* Si el proceso **termina antes** de que finalice el quantum → **libera la CPU voluntariamente**.
* Si el proceso **no termina** cuando finaliza el quantum → el planificador lo **desaloja** y lo **recoloca al final** de la cola de listos.

### Ventajas:

* Buena **equidad**: todos los procesos obtienen acceso a la CPU de forma pareja.
* Mejora la **respuesta para usuarios interactivos**.

### Desventajas:

* Si el **quantum es muy grande** → se parece a FCFS.
* Si el **quantum es muy chico** → demasiados cambios de contexto, lo que **reduce la eficiencia**.

