

***C.F.G.S. DESARROLLO DE APLICACIONES MULTIPLATAFORMA***

**MÓDULO:**

**Sistemas Informáticos**

## **Unidad 3**

### **SISTEMAS OPERATIVOS (I)**

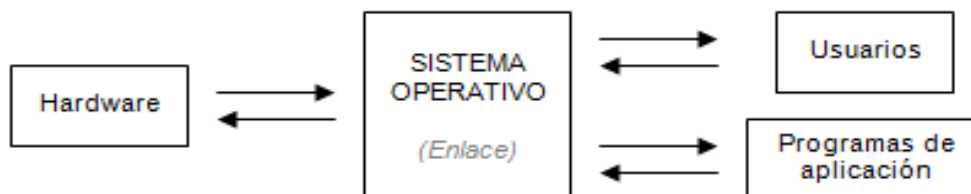
## INDICE DE CONTENIDOS

<b>1. ARQUITECTURA DE UN SISTEMA OPERATIVO.....</b>	<b>4</b>
1.1. DIAGNOSTICO POST .....	4
1.2. CARACTERÍSTICAS.....	5
<b>2. COMPONENTES DE UN SISTEMA OPERATIVO .....</b>	<b>5</b>
<b>3. ESTRUCTURA DE UN SISTEMA OPERATIVO.....</b>	<b>6</b>
<b>4. FUNCIONES DE UN SISTEMA OPERATIVO .....</b>	<b>8</b>
4.1GESTIÓN DE PROCESOS ( $\cong$ Programas $\cong$ Tareas) .....	8
4.1.1 ESTADOS DE UN PROCESO.....	10
4.1.2 PRIORIDADES Y PLANIFICACIÓN DE PROCESOS.....	11
4.1.3 ALGORITMOS DE PLANIFICACIÓN DE PROCESOS .....	12
4.1.3.1 FCFS “Firts-Come, First-Served” .....	12
4.1.3.2 SJF “Shortest Job First” .....	13
4.1.3.3 SRTF “Short Remaining Time First” .....	14
4.1.3.4 Round Robin.....	15
4.1.3.5 Algoritmo por prioridades o multinivel.....	16
4.2GESTIÓN DE MEMORIA.....	16
4.2.1 GESTIÓN DE MEMORIA EN SISTEMAS MONOTAREA.....	16
4.2.2 GESTIÓN DE MEMORIA EN SISTEMAS MULTITAREA .....	17
4.2.2.1 Redistribución de memoria.....	18
4.2.2.2 Paginación .....	20
4.2.2.3 Segmentación.....	21
4.2.2.4 Memoria virtual.....	22
4.3GESTIÓN DE ENTRADAS Y SALIDAS .....	24
4.3.1 CONTROLADORES DE DISPOSITIVOS ( $\cong$ Device drivers) .....	24
4.3.2 MÉTODOS DE FUNCIONAMIENTO DE LOS CONTROLADORES.....	25
4.3.2.1 ENTRADA/SALIDA PROGRAMADA.....	25
4.3.2.2 ENTRADA/SALIDA POR INTERRUPCION .....	26
4.3.2.3 ACCESO DIRECTO A MEMORIA (DMA -Direct Memory Access).....	26
4.4ESTRUCTURAS DE DATOS USADAS EN LA ENTRADA/SALIDA .....	26
4.4.1 SPOOLS.....	27
4.4.2 BUFFERS.....	27
4.5 GESTIÓN DE DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO: DISCOS DUROS.....	28
4.5.1 ALGORITMOS PARA GESTIONAR LAS PETICIONES DE ACCESO A DISCO.....	29
4.5.1.1 Algoritmo FCFS ( $\cong$ First Come, First Served) .....	29
4.5.1.2 Algoritmo SSF ( $\cong$ Shortest Seek First).....	30
4.5.1.3 Algoritmo SCAN o ALGORITMO DEL ASCENSOR .....	30
4.5.1.4 Algoritmo C-SCAN O ALGORITMO SCAN CIRCULAR .....	31
4.6GESTIÓN DE ARCHIVOS Y DISPOSITIVOS.....	32
4.7GESTIÓN DE LA RED .....	32
<b>5. EVOLUCIÓN HISTÓRICA.....</b>	<b>32</b>

5.1 PRIMERA GENERACION (1945-1955).....	32
5.2 SEGUNDA GENERACION (1955-1965).....	33
5.3 TERCERA GENERACION (1965-1980) .....	34
5.4 CUARTA GENERACION (1980 - hasta nuestros días).....	34
<b>6. TIPOS DE SISTEMAS OPERATIVOS .....</b>	<b>35</b>
6.1 SEGÚN EL NÚMERO DE USUARIOS .....	36
6.1.1 MONOUSUARIO .....	36
6.1.2 MUTIUSUARIO .....	37
6.2 SEGÚN EL NÚMERO DE TAREAS (≡ PROCESOS ≡ ventanas) .....	39
6.2.1 MONOTAREA (≡ monoprogramación).....	39
6.2.2 MULTITAREA (≡ multiprogramación).....	40
6.3 SEGÚN EL NÚMERO DE PROCESADORES (≡ Microprocesadores) .....	40
6.3.1 MONOPROCESADOR (≡ MONOPROCESO).....	40
6.3.2 MULTIPROCESADOR (≡ MULTIPROCESO) .....	41
6.4 SEGÚN EL TIEMPO DE RESPUESTA.....	41
6.4.1 PROCESOS POR LOTES (BATCH) .....	41
6.4.2 TIEMPO REAL (≡ REAL TIME).....	41
6.4.3 TIEMPO COMPARTIDO.....	42
6.5 SEGÚN SU DISPONIBILIDAD .....	42
6.5.1 PROPIETARIOS.....	42
6.5.2 LIBRES .....	43
<b>7. TIPOS DE APLICACIONES (≡ SOFTWARE) .....</b>	<b>43</b>
7.1 GRATUITO (≡ freeware) O COMERCIAL .....	43
7.1.1 SOFTWARE GRATUITO .....	43
7.1.2 SOFTWARE COMERCIAL.....	44
7.2 LIBRE O PROPIETARIO .....	44
7.2.1 SOFTWARE LIBRE (≡ free software (no confundir con software gratis) ≡ software de código abierto).....	44
7.2.2 SOFTWARE PROPIETARIO (≡ software no libre ≡ software de código cerrado) .....	45
7.3 OPENSOURCE O PRIVATIVO.....	45
7.3.1 SOFTWARE OPENSOURCE.....	45
7.3.2 SOFTWARE PRIVATIVO.....	45
<b>8. TIPOS DE LICENCIAS .....</b>	<b>45</b>
8.1 OEM .....	45
8.2 RETAIL .....	45
8.3 LICENCIAS POR VOLUMEN .....	45
<b>9. GESTORES DE ARRANQUE .....</b>	<b>46</b>
9.1 NTLDR (≡ NT Loader) .....	46
9.2 BOOT MANAGER (≡ BOOTMGR ≡ Administrador de arranque de Windows).....	47
9.3 LILO (≡ Linux Loader) .....	47
9.4 GRUB .....	48
9.5 GESTORES DE ARRANQUE DE WINDOWS.....	48
9.6 GRUB Y LILO, LOS GESTORES DE ARRANQUE DE LINUX .....	48
9.7 GESTORES DE ARRANQUE INDEPENDIENTES .....	49
9.7.1 GESTOR DE ARRANQUE BOOTIT .....	49
9.7.2 GAG, EL GESTOR DE ARRANQUE GRAFICO .....	50

## 1. ARQUITECTURA DE UN SISTEMA OPERATIVO

El sistema operativo hace de enlace entre el hardware de nuestra máquina y los programas de aplicación que utilizemos:



Vamos a ver a continuación algunas definiciones de sistema operativo. Puedes tener una idea bastante completa de lo que estamos estudiando si combinas todas estas definiciones.

Un **Sistema Operativo (SO)** es un conjunto de programas y funciones que gestionan y coordinan el funcionamiento del hardware y software, ofreciendo al usuario una forma sencilla de comunicarse con el ordenador.

Puedes imaginar un sistema operativo como los programas que hacen utilizable el hardware. El hardware proporciona la "capacidad bruta de proceso"; los sistemas operativos ponen dicha capacidad de proceso al alcance de los usuarios y administran cuidadosamente el hardware para lograr un buen rendimiento.

Los sistemas operativos son ante todo **administradores de recursos**; el principal recurso que administran es el hardware del ordenador.

**Un sistema operativo es un programa que actúa como intermediario entre el usuario y el hardware del ordenador y su propósito es proporcionar el entorno en el cual el usuario pueda ejecutar programas.** Entonces, el objetivo principal de un sistema operativo es, lograr que el sistema informático se use de manera cómoda, y el objetivo secundario es que el hardware del computador se emplee de manera eficiente.

### 1.1. DIAGNOSTICO POST

Cuando se conecta el ordenador **se carga parte del sistema operativo en la memoria y se ejecuta.** El sistema operativo "despierta" al ordenador y hace que reconozca al microprocesador, la memoria, las unidades de disco y cualquier otro dispositivo conectado a ella como el teclado, el ratón, la impresora,... **verificando así que no existan errores de conexión y que todos los dispositivos se han reconocido y trabajan correctamente. A este primer diagnóstico se le denomina POST.**

## 1.2. CARACTERÍSTICAS

Las características generales de los sistemas operativos son las siguientes:

**Concurrencia.** Consiste en la existencia de varias actividades simultáneas y su solución.

**Compartición de recursos.** Las principales razones para permitir la compartición de recursos son la reducción de coste, reutilización, compartición de datos, y eliminación de redundancia.

**Almacenamiento a largo plazo.** Para un almacenamiento de los datos a plazo largo se utilizarán medios no volátiles. Ejm – Disco duro, disquete, CD\_ROM, DVD\_ROM, ...

**Indeterminismo.** El sistema operativo puede ser:

- **Determinista** – Un mismo programa ejecutado en momentos diferentes produce los mismos resultados.
- **Indeterminista** – Tiene que responder a circunstancias que pueden ocurrir en un orden impredecible.

**Eficiencia.** El sistema operativo tiene que ejecutar rápidamente los procesos y optimizar la utilización de recursos.

**Fiabilidad.** El sistema operativo tiene que estar libre de errores.

**Facilidad de corrección.** El sistema operativo tiene que ofrecer modularidad, buenas interfaces y documentación suficiente para facilitar las tareas, para mejorar las prestaciones o para corregir errores. Ejm – Ayuda de Windows.

**Tamaño reducido.** Cuanta menos memoria utilice el sistema operativo, será más eficiente.

## 1. COMPONENTES DE UN SISTEMA OPERATIVO

El sistema operativo proporciona un **entorno en el cual se ejecutan los programas**. Dicho entorno **divide lógicamente el sistema en pequeños módulos creando un interfaz bien definido para los programas que se ejecutarán**. A continuación revisamos algunos de los componentes que debe incluir todo sistema operativo:

**El núcleo.** Transforma los recursos reales (Impresora) del ordenador en recursos estándares (El S.O. se encarga de hacer la comunicación con la impresora) y cómodos de usar.

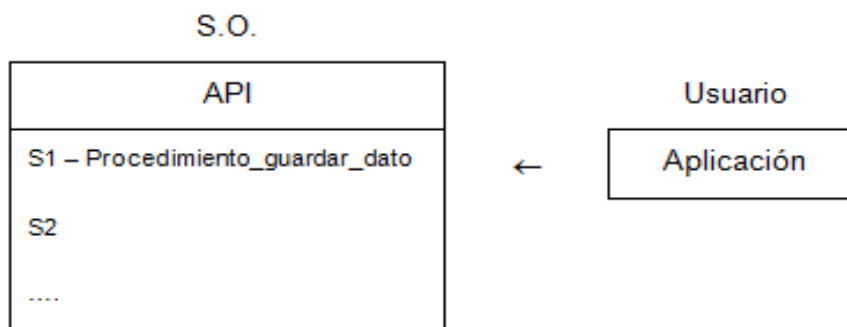
Sus funciones son:

- Controlar las interrupciones
- Gestionar las E/S

- Gestionar los procesos
- Sincronización de procesos

**La API ( $\cong$  Interfaz de Programación de Aplicaciones) del núcleo.** Es el conjunto de servicios que ofrece el sistema a las aplicaciones de usuarios de ese sistema. Las aplicaciones invocan estos servicios a través de llamadas a procedimientos. La API queda definida por lo tanto por los nombres de estos procedimientos, sus argumentos y el significado de cada uno de ellos.

Si por ejemplo el usuario ordena guardar un dato en memoria, se llama al procedimiento "procedimiento\_guardar\_dato" – que indica los pasos a seguir para guardar ese dato en memoria.



El **conjunto de servicios que ofrece el núcleo a los procesos** se denomina la **API del núcleo**. Está formada por procedimientos pertenecientes al núcleo, pero que se invocan desde un proceso cualquiera. La invocación de uno de estos procedimientos es una llamada al sistema.

**Los drivers para dispositivos.** Un driver o controlador hace que el sistema operativo de una computadora pueda entenderse con cualquier periférico, como es el caso de una impresora, una placa de video, un mouse, un módem, etc.

Un driver es como un traductor entre el ordenador y el periférico.

**El sistema de archivos.** Se encarga de estructurar un disco en una estructura jerárquica de archivos y directorios.

**El intérprete de comandos ( $\cong$  shell).** Se encarga de leer las órdenes interactivas de usuario y ejecutar los programas que el usuario indique.

## 2. ESTRUCTURA DE UN SISTEMA OPERATIVO

**Los sistemas operativos se organizan en capas o niveles en torno a un núcleo principal.** Cada una de estas capas o niveles realiza una función determinada y, dependiendo de esa función, tienen más o menos prioridad. La capa principal y la de mayor prioridad es el núcleo.

Un sistema operativo se puede estructurar o dividir en cuatro capas o niveles:

- **Nivel usuario.** Recoge las órdenes que el usuario da al ordenador.
- **Nivel supervisor.** Se encarga de realizar la comunicación de cada proceso entre el sistema y el usuario. O sea **comprueba las órdenes que da el usuario.**

Controla y coordina la gestión de entrada/salida de los diferentes procesos hacia los periféricos.

- **Nivel ejecutivo.** Realiza la administración y gestión de la memoria. Se encarga de almacenar los procesos en páginas, tanto en memoria principal como en disco. Ya veremos que esta gestión es la llamada "**gestión de memoria virtual**".
- **Nivel núcleo.** Es el que se encarga de controlar todo lo que ocurre en el ordenador. Gestionan los procesos que llegan para ser ejecutados. Son fundamentalmente, sistemas operativos multiusuario. **Este nivel se encarga de realizar tareas básicas del sistema, comunicación con el hardware, planificación de procesos, etc.**

Las capas de un sistema operativo, relacionadas con los niveles, son las siguientes:

CAPA	NIVEL
3	Usuario
2	Supervisor
1	Ejecutivo
0	Núcleo

Dependiendo del SO que tengamos instalado este va a tener más o menos niveles.

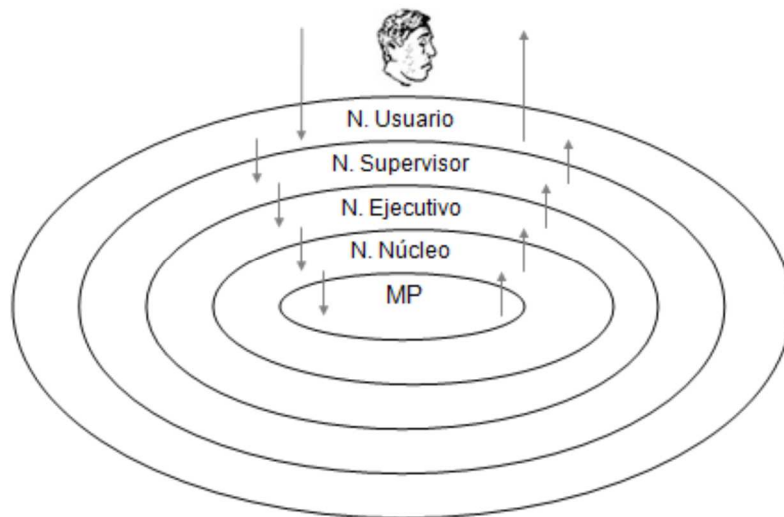
**Ejemplo:** Vamos a ver cómo funcionan los niveles del SO en el caso de que el usuario ordene poner en negrita un texto:

- Esa orden se pasa al "Nivel de usuario" y el nivel de usuario *recoge* la orden.
- Lo pasa al "Nivel supervisor" y el supervisor *comprueba* que esa orden es correcta.

Ejemplo de orden incorrecta – No haber seleccionado el texto, entonces no puede cumplir la orden porque no tiene nada que poner en negrita.

- Se pasa al “Nivel ejecutivo” y el nivel ejecutivo selecciona ese texto en *memoria* y lo prepara para ponerlo en negrita.
- El “núcleo” le comunica al microprocesador que tiene que mostrar en pantalla el texto en negrita.
- El “microprocesador” – *Ejecuta* la orden, o sea pone el texto en negrita. Esa orden consta de dos partes:
  - Mostrar el texto (en negrita) en pantalla.
  - Y almacenarlo en memoria ya puesto en negrita.
- El resultado lo pasa al Núcleo, este al N. Ejecutivo, este al N. Supervisor, este al N. Usuario, que muestra al usuario el texto en negrita

Esquemáticamente:



### 3. FUNCIONES DE UN SISTEMA OPERATIVO

#### 3.1 GESTIÓN DE PROCESOS ( $\cong$ Programas $\cong$ Tareas)

En la actualidad, cualquier ordenador realiza varias cosas al mismo tiempo, ya que la mayoría de los sistemas operativos actuales son multitarea. Los únicos sistemas multitarea real son aquellos que cuentan con más de un procesador. El resto son sistemas operativos que permiten cargar en memoria más de un proceso, pero solamente pueden ejecutar uno de ellos



simultáneamente. Además, puede ser que mientras que se ejecuta un programa se estén enviando datos a la impresora o se estén admitiendo datos por el teclado.

Este paralelismo de tareas necesita una planificación especial para optimizar el uso de los recursos del sistema.

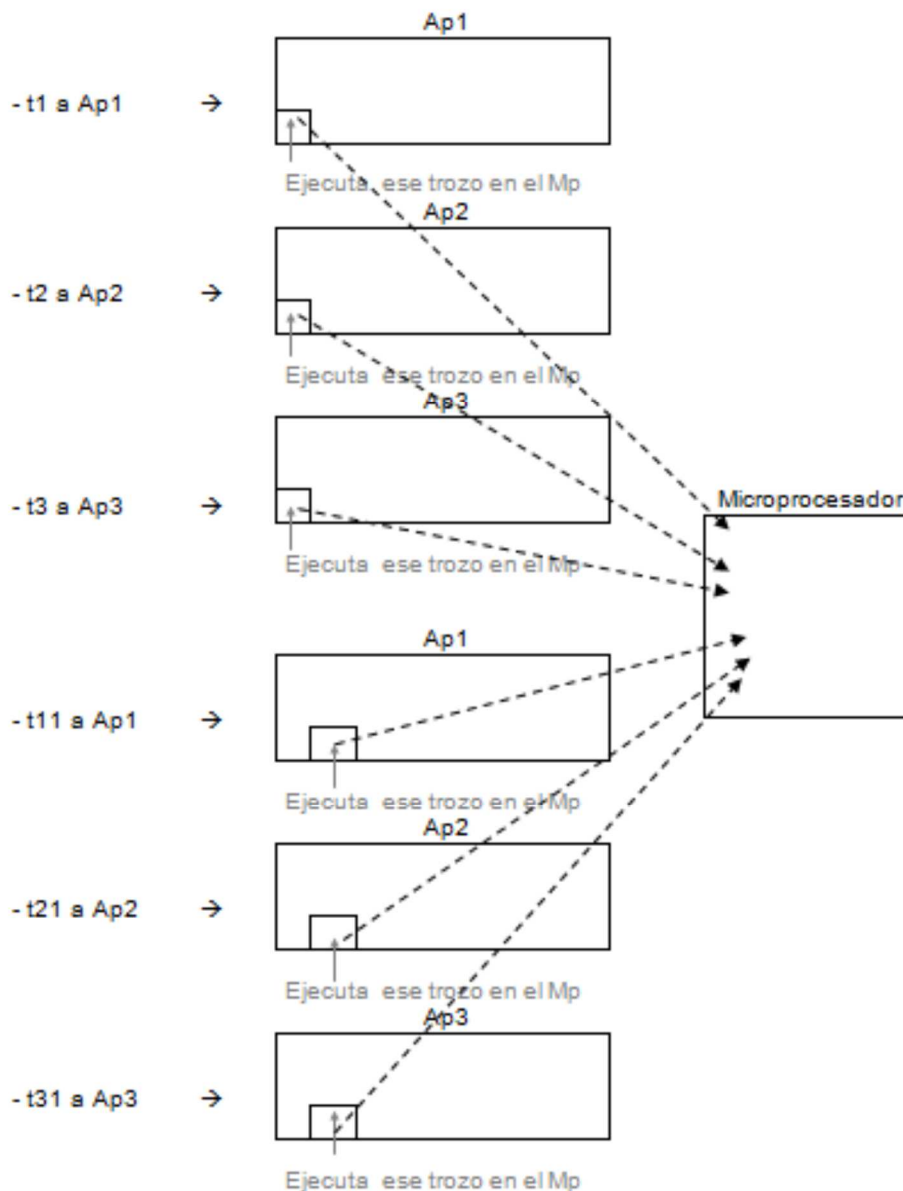
**Un proceso** – Es un programa en ejecución. Pero si tenemos en cuenta la multitarea y el mutiproceto, hemos de considerar que varios de estos programas podrán estar ejecutándose a la vez, y, para ello, el sistema operativo tendrá que llevar a cabo una adecuada gestión de los recursos físicos del propio sistema informático.

Una CPU no puede realizar dos o más procesos a la vez. La rapidez con la que la UCP dedica de forma alterna su tiempo a los diferentes procesos puede inducir a pensar que los procesos se ejecutan simultáneamente, pero no es así: la UCP dedica su tiempo en fracciones de segundo a cada proceso, pero no de forma simultánea.

La mayoría de los ordenadores tienen un único microprocesador (Mp)  $\Rightarrow$  Cuando se ejecutan (realizan) varios procesos (tareas) a la vez hay que compartir el tiempo del Mp. Que consiste en dividir el tiempo del Mp en intervalos de tiempo (milisegundos) y asignar ese intervalo de tiempo a cada proceso que se está ejecutando. De esta forma, el procesador trabajará poco tiempo en cada proceso.

Ejemplo: Si se ejecutan (lanzan) 3 aplicaciones ( $\cong$  tareas  $\cong$  programas  $\cong$  procesos): Ap1, Ap2, Ap3, el Microprocesador no puede ejecutar las 3 a la vez  $\Rightarrow$

1. Primero asigna un tiempo  $t_1$  a Ap1 y ejecuta un trozo de  $t_1$  en el microprocesador.
2. Luego asigna un tiempo  $t_2$  a Ap2 y ejecuta un trozo de  $t_2$  en el microprocesador.
3. Luego asigna un tiempo  $t_3$  a Ap3 y ejecuta un trozo de  $t_3$  en el .
4. Y vuelve otra vez a repetir los puntos 1. 2. 3. Hasta que vaya terminando de ejecutar las aplicaciones.



#### 4.1.1 ESTADOS DE UN PROCESO

Los estados en los que se puede encontrar un proceso son los siguientes:

**Nuevo.** Proceso que se acaba de crear pero que todavía no ha sido admitido por el SO en el grupo de procesos ejecutables.

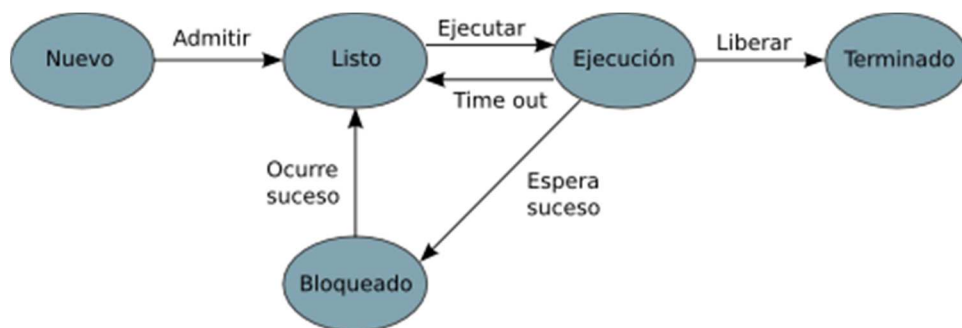
**Preparado (listo).** Un proceso está preparado para ser ejecutado en cuanto se le dé la oportunidad; es decir, está esperando el turno para poder utilizar su intervalo de tiempo.

**En ejecución.** El procesador está ejecutando instrucciones de ese proceso en un instante concreto.

**Bloqueado.** El proceso está retenido; es decir, está bloqueado debido a causas múltiples:

- Que dos procesos utilicen el mismo fichero de datos.
- Que dos procesos necesiten utilizar la misma unidad de CD-ROM para cargar determinados datos
- .....

**Terminado.** Proceso que ha sido excluido por el sistema operativo del grupo de procesos ejecutables.



#### 4.1.2 PRIORIDADES Y PLANIFICACIÓN DE PROCESOS

**Prioridades.** Son aquellas que el administrador del sistema asigna a cada proceso. De ello dependerá que un proceso se ejecute en más o menos tiempo.

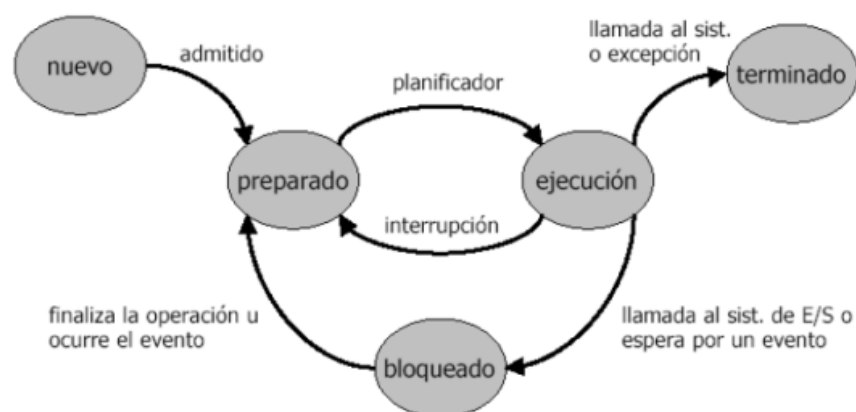
Ejemplo - Supongamos una aplicación informática bancaria que consta de diferentes programas. Es evidente que algunos de estos programas tendrán prioridad sobre otros. Por ejemplo, un programa de realización de nóminas tendrá prioridad de ejecución en los últimos días del mes respecto a un programa de contabilidad.

También se pueden establecer prioridades en función de la necesidad de ejecución de algunos programas. Los programas que más se ejecutan, o sea, los más necesarios, tendrán más prioridad sobre aquellos que se ejecutan muy de cuando en cuando.

**Planificación.** Indica al ordenador los procesos que deben ejecutarse y en qué orden ⇒ Aparecen los algoritmos de planificación:

### 4.1.3 ALGORITMOS DE PLANIFICACIÓN DE PROCESOS

Aquí vamos a centrarnos en analizar los distintos tipos de algoritmos de planificación. Estos algoritmos surgen debido a la necesidad de poder organizar los procesos de una manera eficiente para el procesador. Los algoritmos de planificación se encargan de asegurar que un proceso no monopoliza el procesador. Un proceso es un programa que está en ejecución. Este proceso puede estar en 3 estados distintos "Listo" "Bloqueado" y "En Ejecución". Los procesos son almacenados en una lista junto con la información que indica en qué estado está el proceso, el tiempo que ha usado el CPU, etc.



Dentro de los algoritmos de planificación, citaremos algunos de los más importantes:

#### 4.1.3.1 FCFS "Firts-Come, First-Served"

En esta política de planificación, el procesador ejecuta cada proceso hasta que termina, por tanto, los procesos que en cola de procesos preparados permanecerán encolados en el orden en que lleguen hasta que les toque su ejecución. Este método se conoce también como **FIFO** (first input, first output, **Primero en llegar primero en salir**).

Se trata de una política muy simple y sencilla de llevar a la práctica, pero muy pobre en cuanto a su comportamiento.

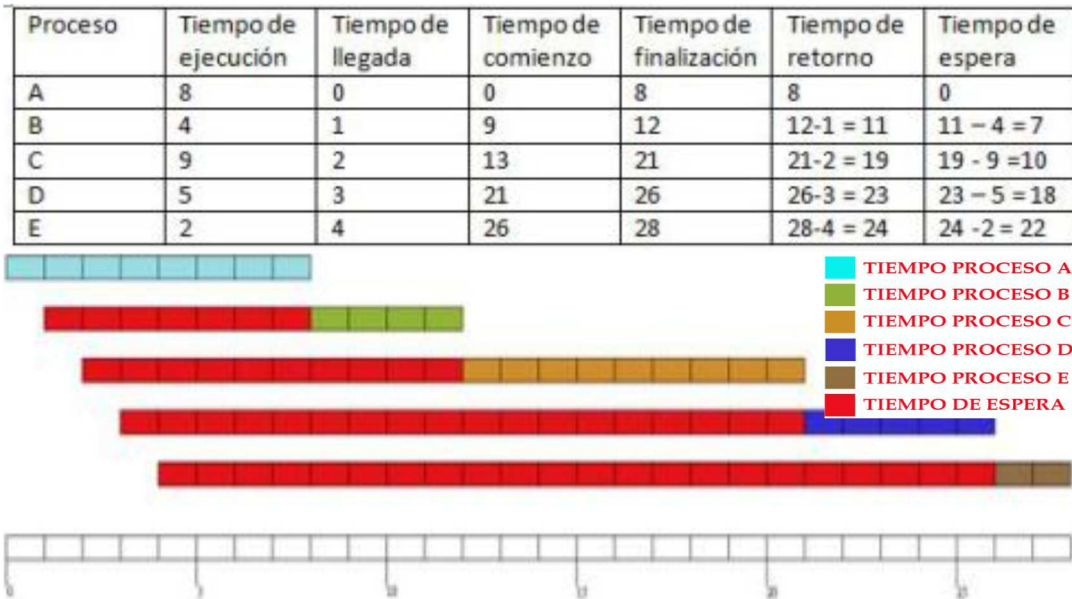
La cantidad de tiempo de espera de cada proceso depende del número de procesos que se encuentren en la cola en el momento de su petición de ejecución y del tiempo que cada uno de ellos tenga en uso al procesador, y es independiente de las necesidades del propio proceso.

Sus características son:

- **No apropiativa.** No hay procesos que se apropien de tiempos de CPU de otros procesos (un proceso se apodera de la CPU hasta que termina de ejecutarse)

- Es **justa**, aunque los procesos largos hacen esperar mucho a los cortos.
- **Predecible**. Sabemos de antemano que el orden de llegada de los procesos es el orden en que se van a ejecutar.
- El tiempo medio de servicio es muy variable en función del número de procesos y su duración.

### Ejemplo:



En el caso de que los procesos de mayor tiempo de duración llegasen los primeros, el tiempo medio de espera sería mucho mayor. Podemos llegar a la conclusión de que este no es un algoritmo eficiente.

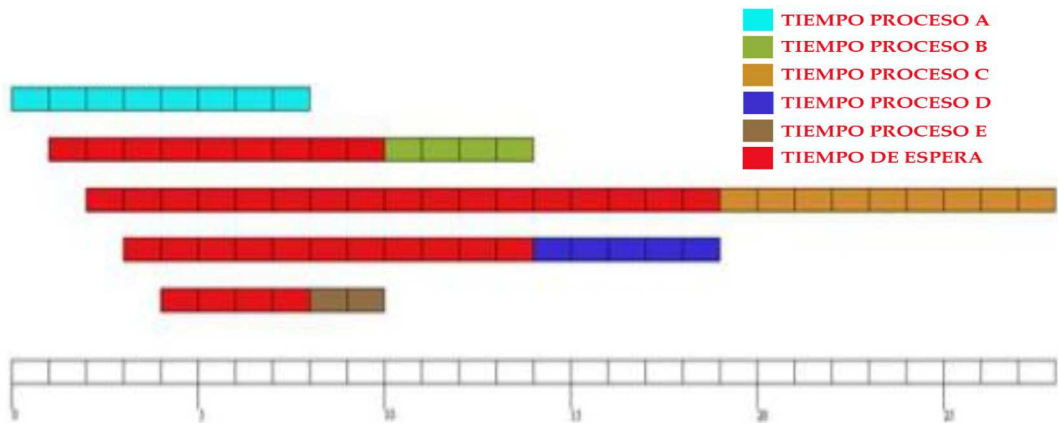
#### 4.1.3.2 SJF “Shortest Job First”

En este algoritmo, da bastante prioridad a los procesos más cortos a la hora de ejecución y los coloca en la cola.

### Ejemplo:

Una cola de personas en Mercadona delante de la caja, la persona que menos compra lleva esa pasa primero.

Proceso	Tiempo de ejecución	Tiempo de llegada	Tiempo de comienzo	Tiempo de finalización	Tiempo de retorno	Tiempo de espera
A	8	0	0	8	8	0
B	4	1	10	14	$14 - 1 = 13$	$13 - 4 = 9$
C	9	2	19	28	$28 - 2 = 26$	$26 - 9 = 17$
D	5	3	14	19	$19 - 3 = 16$	$16 - 5 = 11$
E	2	4	8	10	$10 - 4 = 6$	$6 - 2 = 4$



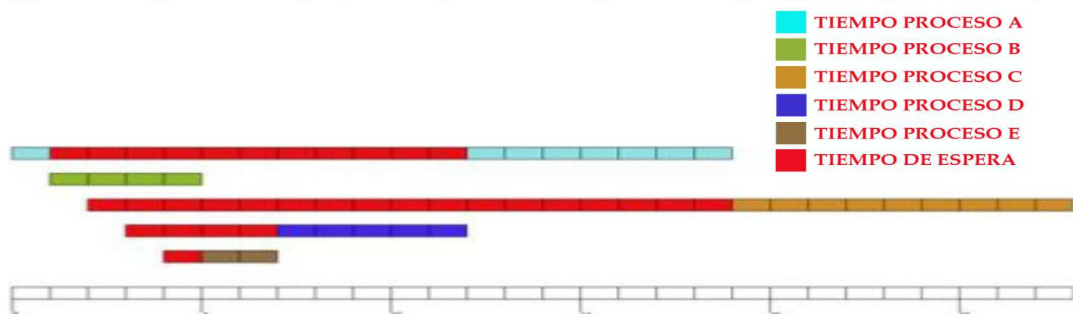
En resumen, este algoritmo selecciona al proceso con el próximo tiempo ejecución más corto (en proceso corto saltará a la cabeza de la cola). Ejecución de un proceso consiste en ciclos de ejecución de CP y ciclos de espera por E/S. El algoritmo selecciona aquel proceso cuyo próximo ciclo de ejecución de CP sea menor. El problema está en conocer dichos valores, pero podemos predecirlos usando la información de los ciclos anteriores ejecutados

#### 4.1.3.3 SRTF “Short Remaining Time First”

Es similar al SJF, con la diferencia de que si un nuevo proceso pasa a listo se activa el dispatcher para ver si es más corto que lo que queda por ejecutar del proceso en ejecución. Si es así, el proceso en ejecución pasa a listo y su tiempo de estimación se decrementa con el tiempo que ha estado ejecutándose.

Los procesos llegan a la cola y solicitan un intervalo de CPU – Si dicho intervalo es inferior al que le falta al proceso en ejecución para abandonar la CPU, el nuevo proceso pasa a la CPU y el que se ejecutaba a la cola de preparados.

Proceso	Tiempo de ejecución	Tiempo de llegada	Tiempo de comienzo	Tiempo de finalización	Tiempo de retorno	Tiempo de espera
A	8	0	0-12	1-19	19	$19 - 8 = 11$
B	4	1	1	5	$5 - 1 = 4$	$4 - 4 = 0$
C	9	2	19	28	$28 - 2 = 26$	$26 - 9 = 17$
D	5	3	7	12	$12 - 3 = 9$	$9 - 5 = 4$
E	2	4	5	7	$7 - 4 = 3$	$3 - 2 = 1$



El intervalo de CPU es difícil de predecir.

–Posibilidad de **inanición**: los trabajos largos no se ejecutarán mientras hayan trabajos cortos.

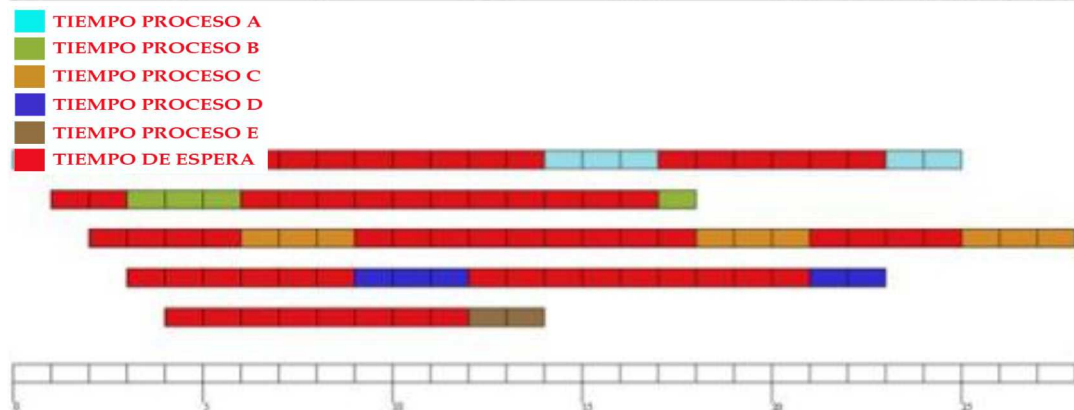
#### 4.1.3.4 Round Robin

Es un método para **seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional**, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.

Round Robin es uno de los algoritmos de planificación de procesos más complejos y difíciles, dentro de un sistema operativo asigna a cada proceso una porción de tiempo equitativa y ordenada, tratando a todos los procesos con la misma prioridad.

Se define un intervalo de tiempo denominado cuanto, cuya duración varía según el sistema. **La cola de procesos se estructura como una cola circular**. El planificado la recorre **asignando a los procesos un tiempo de ejecución llamado quantum**. La organización de la cola es **FIFO** (primero en entrar primero en salir).

Proceso	Tiempo de ejecución	Tiempo de llegada	Tiempo de comienzo	Tiempo de finalización	Tiempo de retorno	Tiempo de espera
A	8	0	0-14 -23	3 -17 -25	25	$25 - 8 = 17$
B	4	1	3 -17	6 -18	$18 - 1 = 17$	$17 - 4 = 13$
C	9	2	6 - 18 - 25	9 -21 -28	$28 - 2 = 26$	$26 - 9 = 17$
D	5	3	9 - 21	12 -23	$23 - 3 = 20$	$20 - 5 = 15$
E	2	4	12	14	$14 - 4 = 4$	$10 - 2 = 2$



Si el proceso agota su quantum de tiempo, se elige a otro proceso para ocupar la CPU. Si el proceso se bloquea o termina antes de agotar su quantum también se alterna el uso de la CPU. El round robín es muy fácil de implementar. Todo lo que necesita el planificado es mantener una lista de los procesos listos.



#### 4.1.3.5 Algoritmo por prioridades o multinivel

Es uno de los más complejos y eficaces. Asigna los tiempos de ejecución de la CPU según una lista de prioridades. Los procesos de mayor prioridad se ejecutan primero. En cada una de las listas, el sistema operativo incluirá aquellos procesos a los que se haya asignado esa prioridad. El tiempo de ejecución del procesador se irá destinando, en primer lugar, de forma secuencial a los procesos de mayor nivel. Terminados éstos, se ejecutarán los procesos del nivel inferior, y así sucesivamente, hasta llegar a los procesos de nivel más bajo.

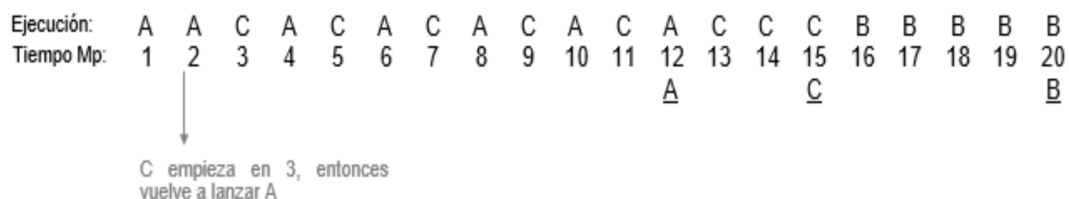
Supongamos que se están ejecutando tres procesos: El A tiene prioridad alta; el B baja; y el C alta. Como A y C tienen prioridad alta entonces se ejecutan primero. B se ejecutará al final.

Supongamos que el primer proceso lanzado es A, el segundo es B y el tercero es C. El resultado es el siguiente:

PROCESOS	Tiempos que consumen de Mp	Tiempo Inicial	Tiempo Final
A	7	1	12
B	5	16	20
C	8	3	15

Lanza: 1º A, 2º B, 3º C

Ejecución:	A	A	C	A	C	A	C	A	C	A	C	A	C	C	C	B	B	B	B	B
Tiempo Mp:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
												A			C					B


  
C empieza en 3, entonces  
vuelve a lanzar A

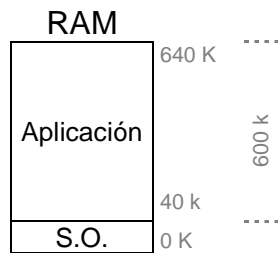
En primer lugar llega el proceso A y como no hay otro proceso se ejecuta. En el siguiente ciclo (ciclo 2) está también disponible el proceso B, pero como el A tiene mayor prioridad continúa ejecutándose. A continuación, en el ciclo 3, tenemos también el proceso C, que tiene la misma prioridad que el A, con lo que saca el proceso A de ejecución y ejecuta el C. A continuación va ejecutando el proceso A y C, alternando ciclos de reloj, hasta que terminan. En el momento en que terminan A y C se ejecuta B.

## 3.2 GESTIÓN DE MEMORIA

### 3.2.1 GESTIÓN DE MEMORIA EN SISTEMAS MONOTAREA

En este tipo de sistemas resulta muy sencilla, una porción de la memoria se reserva al S.O. y el resto se dispone para el programa que se ejecuta.



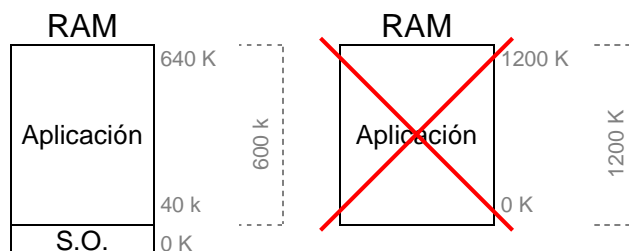


Cuando el programa termina de ejecutarse, se carga otro programa en la misma posición que ocupaba el programa anterior. Como el S.O. tiene un tamaño fijo, los programas se suelen cargar a partir de la siguiente posición libre.

**Inconvenientes** - Este sistema no resulta complicado pero tiene dos inconvenientes:

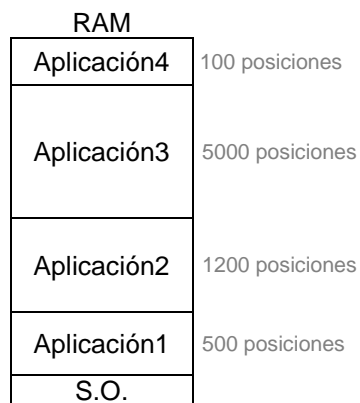
- Solo sirve para sistemas monousuarios, lo que implica una infrutilización de la UCP y los periféricos.
- Es similar a todos los sistemas excepto el de memoria virtual, y consiste en que si el tamaño de un programa es mayor que el que queda libre no puede ejecutarse por falta de memoria.

Por ejemplo: Si quiero ejecutar un programa de 1200 K, no se puede cargar en la RAM, pues es mayor de 640 K – 40 K (lo que ocupa el S.O) = 600 K  
 $\Rightarrow 1200 \text{ k} > 600 \text{ k}$



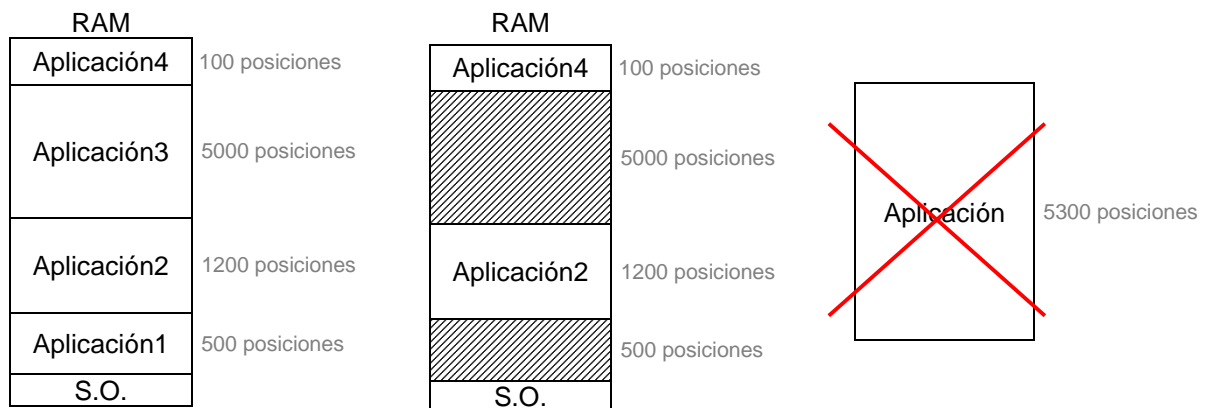
### 3.2.2 GESTIÓN DE MEMORIA EN SISTEMAS MULTITAREA

Con la introducción de los **sistemas Multiusuarios y/o Multitarea**, la gestión de memoria se complica debido a que todo el espacio que no ocupa el S.O. se ha de **repartir entre los diversos programas que están en memoria en un momento dado**.  $\Rightarrow$  Una porción de memoria se reserva al S.O. y el resto se divide en trozos y se asigna a los programas que se van a ejecutar (Aplicación1, Aplicación2, Aplicación3, Aplicación4).



El S.O. tiene una tabla en la que guarda información sobre la memoria que se halla ocupada y la que queda libre en el sistema. Este sistema es relativamente sencillo de implantar, cuando un programa finaliza se actualiza la tabla indicando las posiciones que quedan libres.

**Inconveniente-** El problema que surge con este esquema es la **fragmentación** de memoria. Supongamos que quedan dos regiones libres, la primera de 500 posiciones y la segunda de 5000. Si un programa que ocupa 5300 posiciones estuviera listo para ejecutarse no podría hacerlo, ya que, aunque hay suficiente memoria libre, no es contigua:



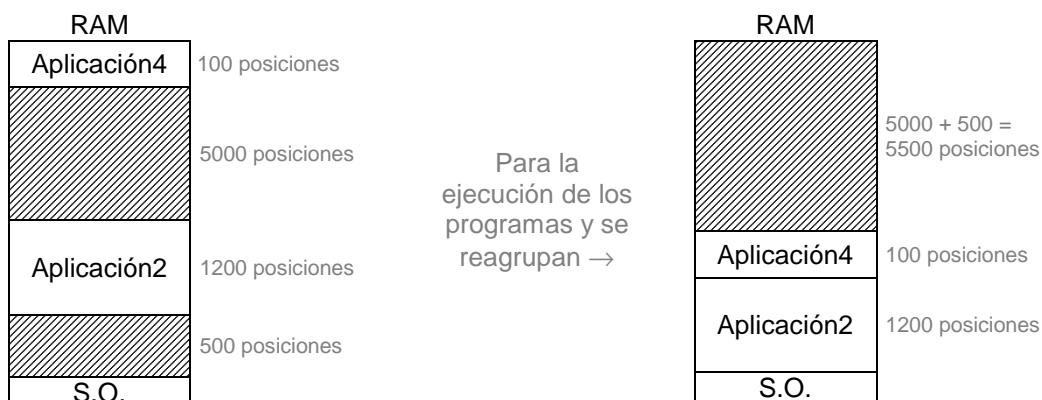
Posiciones  $\cong$  posiciones de memoria  $\cong$  Kbytes

Según se vayan ejecutando programas, la memoria se puede ir fragmentando más y más, quedando una serie de "agujeros" que son porciones muy pequeñas de memoria que no pueden ser utilizadas. Este problema se puede resolver mediante las técnicas: **Redistribución, Paginación, Segmentación y Memoria virtual**:

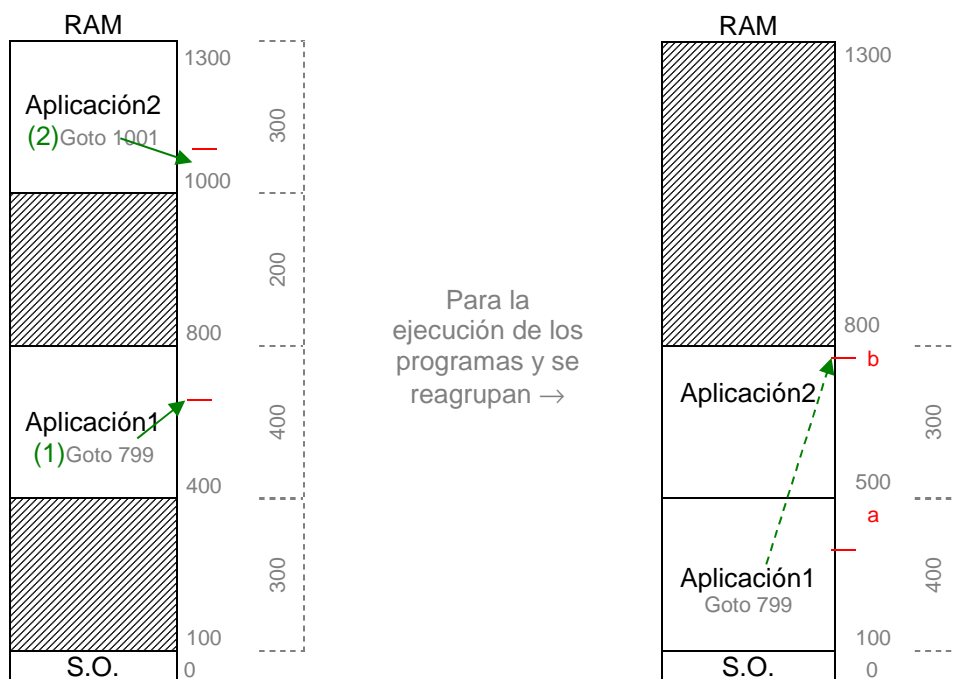
### 3.2.2.1 Redistribución de memoria

Llegado un punto en que la memoria está muy fragmentada, se para la ejecución de los programas y se redistribuye (reagrupa) la memoria, para que todas las posiciones libres se queden en una zona contigua

única. De este modo toda la memoria libre queda disponible en un bloque  $\Rightarrow$  Ya se podría cargar el programa de 5300 posiciones.



**Inconveniente-** Los compiladores producen códigos como si el programa que está traduciendo fuera a ejecutarse a partir de la posición 0. El Loader (programa que carga este del disco a la memoria antes de ejecutarse), una vez que el S.O le ha indicado la posición real que ocupara el programa, se encarga de modificar las direcciones para que el programa se ejecute correctamente en el lugar indicado. Si se redistribuye la memoria y hay instrucciones con saltos, los saltos ya no valen:



Goto 799  $\cong$  Ir a la dirección de memoria 799.

Si Aplicación1 tiene la instrucción goto 799. Iría a "b" en vez de ir a "a", que es donde debería ir.

Este problema de los saltos (goto) se resuelve mediante los **programas reubicables** – Que recalculan el salto. Estos programas usan el llamado

**Registro Base** ( $\cong$  R.B.), que es un registro especial de la UCP que se suma a la dirección computada en la instrucción para obtener la dirección real. Cuando el programa se carga en memoria el registro base se inicializa a cero, sin embargo al redistribuir la memoria y cambiar el programa de lugar, se modifica el registro base para reflejar la nueva situación:

(1) Registro base =  $400 - 100 = +300 \cong$  Lo que ocupa ese programa

$\Rightarrow$  Goto  $\underbrace{799 - 300}_{499} \Rightarrow$  Goto 499  $\Rightarrow$  Ahora si que saltaría a "a"

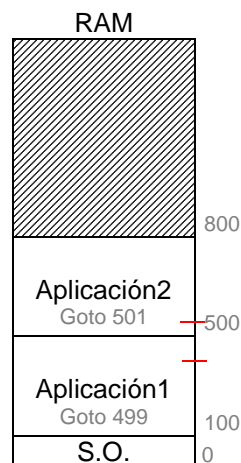
(2) Registro base =

$1000 - 800 = 200$

$400 - 100 = 300$

500

$\Rightarrow$  Goto  $\underbrace{1001 - 500}_{501} \Rightarrow$  Goto 501



**Inconveniente** - Con la redistribución se pierde tiempo. Entonces se usó la paginación.

### 3.2.2.2 Paginación

Consiste en **dividir la memoria en páginas iguales** (del mismo tamaño) y asignarlas (no necesariamente de una forma contigua) a los distintos programas. Una página es la porción mínima de memoria que se puede asignar a un programa, y de ahí hasta el total que necesite.

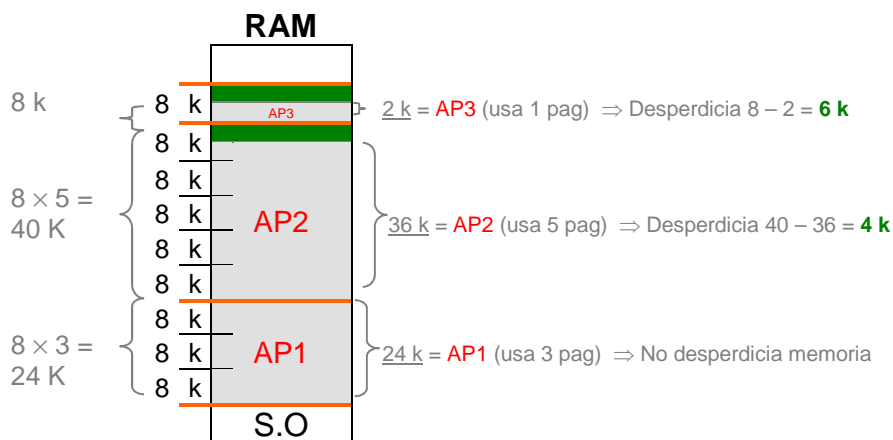
O sea la paginación consiste en dividir la memoria RAM en zonas iguales, llamadas **frames** y los programas en partes del mismo tamaño (que los frames), denominadas **páginas**.

El S.O. contiene una tabla en la que se refleja el estado de las páginas del sistema, es decir, que páginas están listas y cuales asignadas a

programas. Cuando se carga un programa, se calcula el número de páginas que necesita y si están libres se le asignan. Las páginas cargadas en memoria no tienen por qué ser contiguas.

Se divide la memoria en páginas de 8 K.

- |     |   |                                                                                                                                                                                                                                                                                 |
|-----|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AP3 | { | <ul style="list-style-type: none"> <li>• Si metemos en memoria un programa de <u>2 K</u> para ejecutarlo<br/> <math>8\text{ K} - 2\text{ K} = 6\text{ K}</math><br/>           Se desperdician 6 K de RAM.</li> </ul>                                                           |
| AP2 | { | <ul style="list-style-type: none"> <li>• Si metemos en memoria un programa de <u>36 K</u> para ejecutarlo<br/> <math>8\text{ K} \times 5\text{ K} = 40\text{ K} \Rightarrow 40\text{ K} - 36\text{ K} = 4\text{ K}</math><br/>           Se desperdician 4 K de RAM.</li> </ul> |



Que coincida el tamaño de la aplicación con el tamaño de las páginas asignadas es muy difícil.

Con este método, cuando se carga en memoria las direcciones del programa no se modifican, por tanto todas las direcciones que se referencian son relativas a la posición 0, como si el programa estuviera en esa posición (**no haciendo falta que sean reubicables**).

Al tiempo de la carga se crea una **tabla de páginas** en la que se refleja la situación real en memoria de las distintas páginas que componen el programa.

**Inconveniente-** Con la paginación de memoria se reduce considerablemente la fragmentación de memoria, aunque no totalmente, esto es debido a que el tamaño del programa no suele ser múltiplo del tamaño de la página y se desperdicia un cierto espacio de la última página, o sea se desperdicia RAM. Entonces se creó la Segmentación.

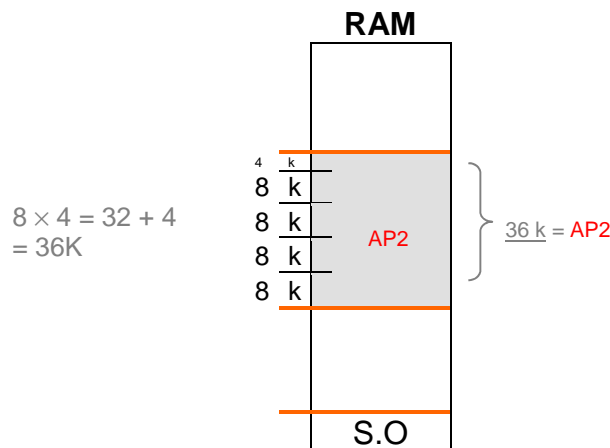
### 3.2.2.3 Segmentación

**Divide la memoria en segmentos**, en vez de páginas.

¿Qué cambia?. **Que los segmentos no tienen que ser todos del mismo tamaño.** ⇒ Los segmentos son de tamaño variable.

Página  $\cong$  Segmento

Si en el ejemplo anterior de la paginación aplicamos la segmentación ⇒ La AP2 ya no desperdiciaría esas 4 K, pues el último segmento en vez de cogerlo de 8 K, lo cogería de 4K.



**Inconveniente de la Segmentación** - Ninguno.

En estas tres técnicas: Redistribución, Paginación, Segmentación. Las aplicaciones se cargan enteras en la RAM, porque antiguamente eran muy pequeñas. Entonces no existía Disco Duro (HD) para intercambiar parte de las aplicaciones.

Como ahora las aplicaciones son cada vez más grandes, es imposible que quepan en RAM. **Por lo que se empezó a utilizar el Disco Duro (HD) como si fuese RAM.** Y a eso lo llama **Memoria Virtual**, aunque básicamente es lo mismo que la paginación.

#### 3.2.2.4 Memoria virtual

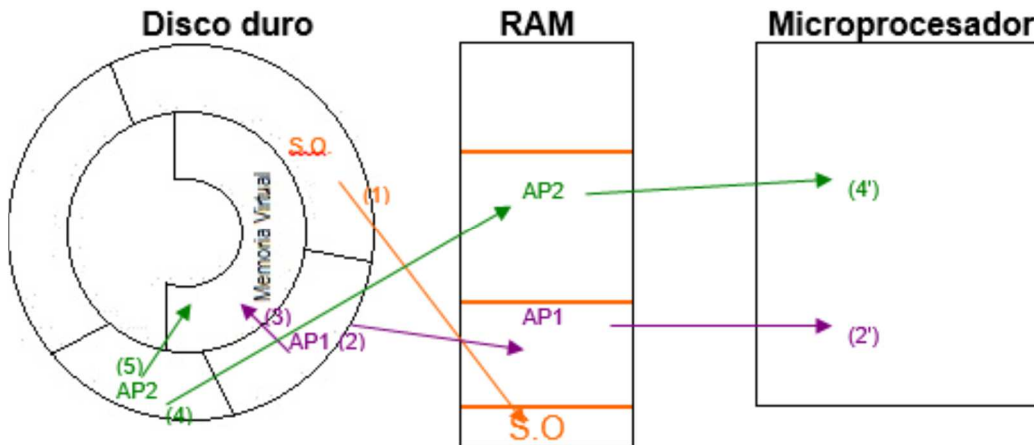
El ultimo método de gestión de memoria que vamos a ver es la de memoria virtual, este **es un método derivado de la paginación** y tiene una ventaja con respecto a los métodos anteriores: **un programa puede ejecutarse aunque su tamaño sea mayor que la memoria disponible.**

En la técnica de Memoria Virtual, el programa no hace falta que este cargado por completo en RAM. Sólo carga (en RAM) la parte de programa que necesita ejecutar, o sea la parte de programa que está usando. Y el resto del programa permanece en disco duro para su posterior utilización, si fuera necesario.

Observamos que durante la ejecución de un programa no es necesario que todo el programa este en memoria. Hay un conjunto de

instrucciones que se repiten una y otra vez durante un tiempo determinado, por ejemplo un bucle, por tanto, no es necesario que la parte del programa que no se está ejecutando se encuentre en memoria. Con la memoria virtual, por cada programa que se ejecuta solo hay un cierto número de páginas en memoria. Este número depende del tamaño del programa y de su prioridad.

**Inconveniente**– Que accede mucho al disco duro (HD) y el acceso al HD es 100 veces más lento que a una RAM.



- Al encender el ordenador el S.O. se pasa a la RAM (1) y se va ejecutando en el Microprocesador. Mientras el ordenador esté encendido, el S.O. siempre va a estar ejecutándose en la RAM.
- El usuario lanza AP1:
  - Se almacena una parte (2) en la RAM (sólo lo que se necesita ejecutar, porque no cabe todo en la RAM)
  - Y el resto (3) quedará en Memoria Virtual
- El usuario lanza AP2:
  - Se almacena una parte (4) en la RAM
  - Y el resto (5) quedará en Memoria Virtual

### **Archivo de paginación ≅ Memoria virtual**

Un archivo de paginación es un área en el disco duro que Windows usa como si fuese RAM. Se puede cambiar el tamaño del archivo de paginación para que la memoria virtual sea de mayor tamaño. Para cambiar el tamaño del archivo de paginación:

**Inicio → Panel de control → Rendimiento y mantenimiento → Sistema → (Ficha) Opciones avanzadas → Rendimiento → Configuración → (Ficha) Opciones avanzadas → Memoria Virtual → Cambiar**

### 3.3 GESTIÓN DE ENTRADAS Y SALIDAS

El ordenador debe incorporar periféricos que facilitan la comunicación de este con el usuario.

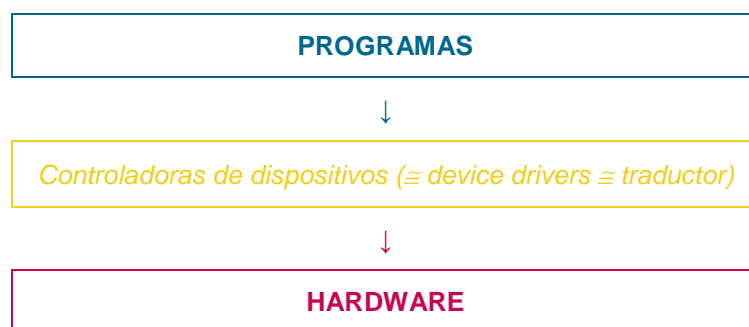
Pues bien, como no podía ser menos el sistema operativo también está detrás de la gestión de estos dispositivos. El sistema operativo gestiona los recursos del ordenador y los dispositivos de entrada/salida que forman parte de los recursos hardware del ordenador. El sistema operativo hace que los dispositivos se conecten al sistema y realicen sus funciones de forma controlada y eficiente. Además se pretende que los programas de aplicación puedan utilizar los recursos hardware a su disposición de una forma unificada, por ejemplo un programa puede escribir y leer datos de un disco duro sin necesidad de conocer el modelo concreto de disco. El sistema operativo debe perseguir que los programas sean independientes de los dispositivos y actúa de intermediario entre ellos.

#### 3.3.1 CONTROLADORES DE DISPOSITIVOS ( $\cong$ Device drivers)

Si existen multitud de dispositivos diferentes en el mercado, de distintos fabricantes. ¿Cómo conseguir que un programa de aplicación pueda entenderse con todos? La respuesta a esta pregunta es qué “Es imposible hacer un programa de aplicación que contemple toda la extensa gama de dispositivos diferentes que se pueden encontrar”.

En lugar de esto lo que se hace es estandarizar el acceso a los dispositivos utilizando lo que se llaman controladores de dispositivos ( $\cong$  device **drivers**). Estos son piezas de software que normalmente son **suministradas por el fabricante del dispositivo** o bien por el fabricante del propio sistema operativo y que acompañan al dispositivo.

Pues bien, estos controladores actúan como traductor ( $\cong$  interface) entre los programas y el hardware.







Otro aspecto a tener en cuenta en el diseño del controlador es si el dispositivo en cuestión admite ser compartido por varios procesos al mismo tiempo o no. Por ejemplo un disco duro puede recibir varias peticiones de procesos a la vez e ir atendíéndolas, pero en cambio una impresora sólo puede procesar un trabajo de impresión en un momento dado, necesitando en este caso que el sistema operativo junto con el controlador formen lo que se llama una cola de impresión para ir imprimiendo los trabajos solicitados.

### 3.3.2 MÉTODOS DE FUNCIONAMIENTO DE LOS CONTROLADORES

Los controladores estudiados anteriormente sirven de puente entre las aplicaciones y los dispositivos de entrada/salida, toman las peticiones de entrada o salida y se ponen en contacto con el dispositivo que controlan para realizar la operación deseada.

Por ejemplo, un ratón como el que estás utilizando ahora mismo genera una serie de impulsos eléctricos cada vez que lo mueves o pulsas uno de sus botones, esos impulsos tienen un reflejo inmediato en el cursor que aparece en pantalla, pues bien es el controlador del ratón el que consigue el efecto. Por eso son tan importantes los controladores.



Existen tres tipos de métodos de funcionamiento de un controlador, según sea el tipo de intervención de la CPU en el proceso:

#### 3.3.2.1 ENTRADA/SALIDA PROGRAMADA

El microprocesador pregunta constantemente a la unidad de E/S si el periférico que quiere está disponible, cuando la respuesta es afirmativa

se inicia la transferencia de información. Por lo tanto, el microprocesador inicia y lleva a cabo la transferencia.

### 3.3.2.2 ENTRADA/SALIDA POR INTERRUPCION

El microprocesador no pregunta a la unidad de E/S por la disponibilidad de un periférico sino que le hace una petición y continúa con el proceso que está realizando. Es la unidad de E/S la encargada de avisar al microprocesador cuando puede iniciar la transferencia. De esta forma el microprocesador no pierde tiempo esperando que el periférico esté disponible. Por lo tanto, el microprocesador ejecuta la transferencia pero el inicio es pedido por el periférico que indica así su disponibilidad.

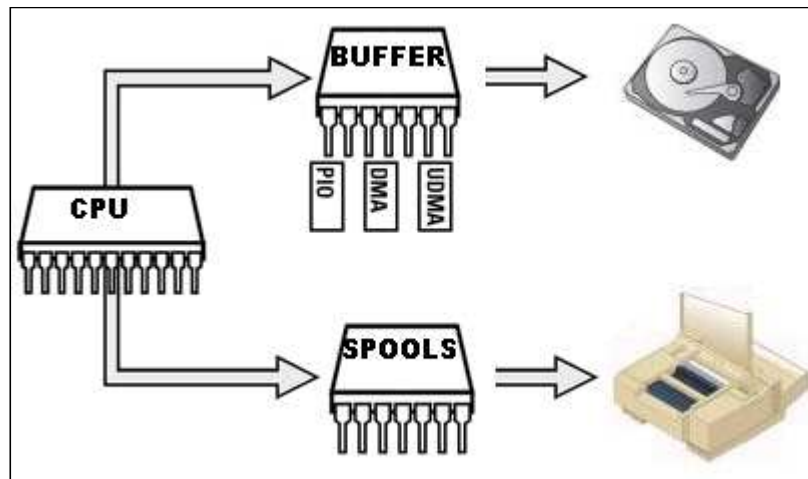
### 3.3.2.3 ACCESO DIRECTO A MEMORIA (DMA -Direct Memory Access)

El **acceso directo a memoria (DMA)**, del inglés direct memory access) permite a cierto tipo de componentes de una computadora acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento (CPU). Muchos sistemas hardware utilizan DMA, incluyendo controladores de unidades de disco, tarjetas gráficas y tarjetas de sonido. DMA es una característica esencial en todos los ordenadores modernos, ya que permite a dispositivos de diferentes velocidades comunicarse sin someter a la CPU a una carga masiva de interrupciones.

Una transferencia DMA consiste principalmente en copiar un bloque de memoria de un dispositivo a otro. En lugar de que la CPU inicie la transferencia, la transferencia se lleva a cabo por el controlador DMA. Un ejemplo típico es mover un bloque de memoria desde una memoria externa a una interna más rápida. Tal operación no ocupa al procesador y, por ende, éste puede efectuar otras tareas. Las transferencias DMA son esenciales para aumentar el rendimiento de aplicaciones que requieran muchos recursos.

## 3.4 ESTRUCTURAS DE DATOS USADAS EN LA ENTRADA/SALIDA

Existen unas estructuras de datos que se utilizan para permitir la comunicación fluida entre dispositivos y CPU. Las más importantes son los spools y los buffers:



### 3.4.1 SPOOLS

Una técnica muy común, especialmente en salida, es el uso de "**spoolers**". Los datos de salida se almacenan de forma temporal en una cola situada en un dispositivo de almacenamiento masivo (**spool**), hasta que el dispositivo periférico requerido se encuentre libre. De este modo se evita que un programa quede retenido porque el periférico no esté disponible. El sistema operativo dispone de llamadas para añadir y eliminar archivos del spool.

Esta técnica **se utiliza en dispositivos que no admiten intercalación**, es decir dispositivos que no pueden atender peticiones de distintos orígenes. Ejemplo – Una impresora. Cuando se empieza a imprimir un trabajo no puede empezar con otro hasta que no haya terminado con el que está.

El dispositivo necesita todos los datos de salida de golpe antes de iniciar su tarea. En el ejemplo una impresora no puede empezar a imprimir si no tiene el fichero que se quiere imprimir entero.

### 3.4.2 BUFFERS

Un buffer es un archivo que reside en memoria.

Ejemplo – Se abre un archivo en Word ⇒ Se crea un buffer en memoria. Cuando se trabaja con el archivo de Word, se trabaja con el archivo que está en memoria (buffer), no con el que está en el disco duro. Al guardar el archivo, se guarda en el disco duro, y desaparece de memoria al cerrarlo.

Cuando estamos con Word, vemos que cada cierto tiempo guarda el archivo con el que estamos trabajando. No lo guarda en el disco duro, en realidad lo está guardando en un 2º Buffer (de archivos temporales) para, el caso de que si se va la luz, poder recuperar ese archivo.

Esta técnica **se utiliza para dispositivos que admiten intercalación**, es decir dispositivos que pueden atender peticiones de distintos orígenes. Ejemplo – El disco duro. Ya que le llegan peticiones de distintos orígenes.

El dispositivo no necesita todos los datos de salida de golpe antes de iniciar su tarea. Pueden enviarse porciones que el buffer retiene de forma temporal.

También **se utilizan para acoplar velocidades de distintos dispositivos**. Como puede ser el caso de un dispositivo lento que va a recibir información más rápido de lo que puede atender, entonces se utiliza un buffer para retener de forma temporal la información hasta que el dispositivo se desahoga un poco. Ejemplo – Si a una grabadora de CD (lenta) el disco duro le envía datos a gran velocidad, entonces se utiliza un buffer para retener esos datos.

### 3.5 GESTIÓN DE DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO: DISCOS DUROS

Normalmente las unidades de disco se configuran automáticamente, pero hay veces en que aparecen problemas por un número de accesos importantes, por lo que es preciso hacer una previsión de tráfico de datos y de almacenamiento, para no superar nunca la capacidad del disco.

Los dispositivos de almacenamiento secundario pueden ser: discos duros, CD, DVD, etc. Nos vamos a centrar en la gestión de los discos duros por ser el elemento principal de esta categoría.



Se puede decir que la velocidad de un ordenador depende en gran medida de lo bien que se gestionen los discos duros por parte del sistema operativo. En las dos primeras unidades del curso estudiaste los aspectos puramente hardware de estos dispositivos, ahora estudiaremos como pueden gestionarse por parte del sistema operativo a través de los controladores de disco duro. En concreto **nos centraremos en los algoritmos que se utilizan para gestionar las peticiones de acceso a disco que realizan los programas de aplicación y el propio sistema operativo**.

Cada unidad de disco, y en general cada dispositivo de E/S, tiene una cola asociada para ir guardando las solicitudes pendientes hasta que puedan ser atendidas. Normalmente los discos sólo pueden atender una petición a la vez (los discos SCSI pueden atender varias concurrentemente) por lo que mientras se procesa una solicitud, pueden llegar otras (pensemos que estamos en un sistema multiusuario y multitarea). Cada solicitud afectará a un cilindro y las cabezas deberán desplazarse a dicho cilindro para procesarla. El objetivo es reducir el tiempo promedio de búsqueda.

### 3.5.1 ALGORITMOS PARA GESTIONAR LAS PETICIONES DE ACCESO A DISCO

Existen 4 algoritmos para gestionar las peticiones de acceso a disco:

#### 3.5.1.1 Algoritmo FCFS ( $\cong$ First Come, First Served)

La **planificación FCFS** (First Come, First Served – Primero en llegar, primero en ser servido) es la planificación más sencilla. Como se desprende de su propio nombre se dará servicio a las solicitudes de acceso a disco de la cola según el orden de llegada de dichas solicitudes. Esta planificación hará uso de una cola tipo FIFO (First In, First Out – Primero en entrar, primero en salir).


Se puede considerar que este algoritmo es evidentemente justo. Sin embargo tiene un inconveniente, en promedio, puede dar lugar a tiempos bastante grandes.

#### EJEMPLO:

Considerar un controlador de disco con la cabeza lectora posicionada en la pista 99 y la dirección de búsqueda creciente. La cola de peticiones es la siguiente:

Peticiones:	81	142	86	172	89	145	97	170	125
-------------	----	-----	----	-----	----	-----	----	-----	-----

Vamos atendiendo las peticiones según nos van llegando. Este algoritmo funciona bien cuando no tenemos demasiadas peticiones:

Disco:	1	81	86	89	97	99	125	142	145	170	172
Orden en el que atiende a las peticiones:		2º	4º	6º	8º		10º	3º	7º	9º	5º

### 3.5.1.2 Algoritmo SSF ( $\cong$ Shortest Seek First)

La **planificación SSF** (Shortest Seek First – Primero la búsqueda más cercana) atiende primero la solicitud de la cola de solicitudes pendientes que quiere acceder al cilindro más cercano al de la solicitud actual, que se está procesando. Es decir, atiende primero la petición que requiere el menor movimiento de la cabeza de lectura/escritura desde su posición actual.

El algoritmo SSF **es un algoritmo bastante habitual**. Un **inconveniente** que aparece es que pueden llegar solicitudes que impliquen cilindros próximos al actual, por lo que estas solicitudes serán atendidas enseguida mientras que otras que llegaron antes, con cilindros más alejados, no se atenderán. Esta situación se conoce con el nombre de **bloqueo indefinido**.


Este algoritmo elige siempre la opción que incurre en el menor tiempo de búsqueda respecto a la posición actual. Sin embargo, **éste tampoco es un algoritmo óptimo**; es decir, no garantiza que la secuencia elegida sea la que menor tiempo promedio de búsqueda tenga.

#### EJEMPLO:

Considerar un controlador de disco con la cabeza lectora posicionada en la pista 99 y la dirección de búsqueda creciente. La cola de peticiones es la siguiente:

Peticiones:	81	142	86	172	89	145	97	170	125
-------------	----	-----	----	-----	----	-----	----	-----	-----

Vamos atendiendo las peticiones que requieren menos tiempo del dispositivo, es decir la que se encuentra más cerca de la petición que se está procesando:

Disco:	1	81	86	89	97	99	125	142	145	170	172
Orden en el que atiende a las peticiones:		5º	4º	3º	2º		6º	7º	8º	9º	10º

### 3.5.1.3 Algoritmo SCAN o ALGORITMO DEL ASCENSOR

La **planificación Scan**, también llamada **algoritmo del ascensor** porque se comporta como tal: va dando servicio a las solicitudes que van encontrando en el sentido en el que se van desplazando las cabezas de lectura/escritura por el disco. Cuando no hay más solicitudes en ese sentido, o se llega al extremo, se invierte el sentido para hacer lo mismo otra vez pero yendo hacia el otro lado. Por tanto, en este

algoritmo es necesario tener un bit que indique el sentido del movimiento.


**Este algoritmo evita el bloqueo indefinido** que se puede producir con la planificación SSF. Una propiedad interesante de este algoritmo es que dada cualquier colección de solicitudes, la cuota máxima del total de movimientos está fijada: es el doble del número de cilindros. En general, **el algoritmo del ascensor es peor que el SSF, aunque es más apropiado para sistemas que hacen gran uso del disco.**

#### EJEMPLO:

Considerar un controlador de disco con la cabeza lectora posicionada en la pista 99 y la dirección de búsqueda creciente. La cola de peticiones es la siguiente:

Peticiones:	81	142	86	172	89	145	97	170	125
-------------	----	-----	----	-----	----	-----	----	-----	-----

El brazo del disco se mueve en un único sentido. Sólo se atenderá la petición más cercana en el sentido en el que estemos recorriendo el disco. Una vez alcanzada la última pista, se sigue el movimiento en sentido opuesto:

Disco:	1	81	86	89	97	99	125	142	145	170	172
Orden en el que atiende a las peticiones:		10°	9°	8°	7°		2°	3°	4°	5°	6°

### 3.5.1.4 Algoritmo C-SCAN O ALGORITMO SCAN CIRCULAR

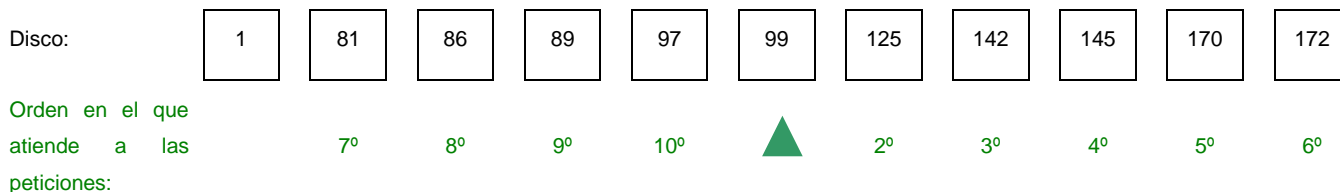
Supongamos una distribución uniforme de solicitudes de pistas. En la planificación Scan, cuando la cabeza llega a un extremo e invierte la dirección, en la zona próxima a dicho extremo habrá pocas solicitudes, ya que las solicitudes para acceder a los cilindros de dicha zona acaban de ser servidas. La mayor densidad de solicitudes se encontrará en el extremo opuesto del disco. **El algoritmo C-Scan o Scan Circular**, trata de evitar el problema anterior restringiendo el rastreo a una única dirección. En esta planificación la cabeza se mueve de un extremo del disco al otro, atendiendo las solicitudes que va encontrando, pero al llegar al extremo opuesto, regresa de inmediato al otro sin servir ninguna solicitud.

#### EJEMPLO:

Considerar un controlador de disco con la cabeza lectora posicionada en la pista 99 y la dirección de búsqueda creciente. La cola de peticiones es la siguiente:

Peticiones:	81	142	86	172	89	145	97	170	125
-------------	----	-----	----	-----	----	-----	----	-----	-----

El brazo del disco se mueve en un único sentido, y de forma circular. Sólo se atenderá la petición más cercana en el sentido en el que estemos recorriendo el disco. Una vez alcanzada la última pista, volvemos a la primera pista.



### 3.6 GESTIÓN DE ARCHIVOS Y DISPOSITIVOS

Se crearán, eliminarán, consultarán y modificarán archivos y directorios.

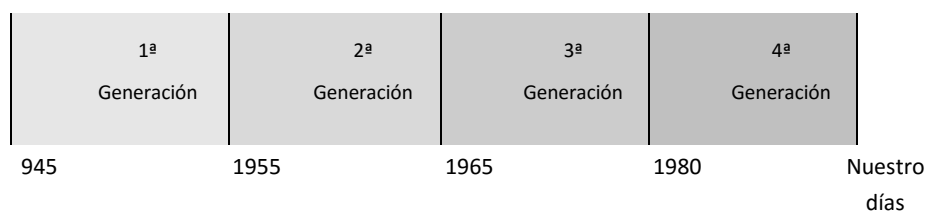
Este punto lo trataremos con más profundidad en el siguiente tema.

### 3.7 GESTIÓN DE LA RED

Se utilizarán tarjetas de red, protocolos de red (TCP/IP,...), aplicaciones para uso de la red (www, FTP,...).

## 4. EVOLUCIÓN HISTÓRICA

Los sistemas operativos, al igual que el hardware, han sufrido cambios a través del tiempo, los cuales se pueden agrupar en generaciones. La evolución del hardware ha marcado el paralelismo de la evolución de los sistemas operativos.



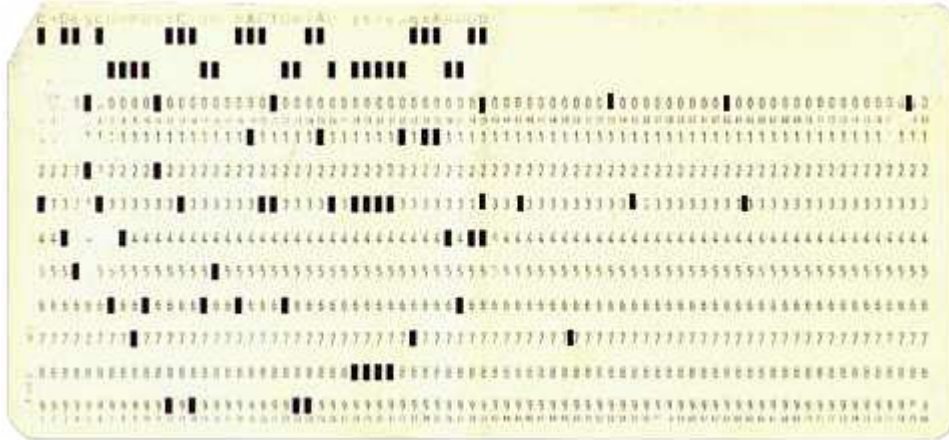
### 4.1 PRIMERA GENERACION (1945-1955)

En esta época se utilizaba la tecnología de las válvulas o tubos de vacío que hacían que los ordenadores fuesen enormes. El usuario tenía que hacer todos los programas en código binario. Las máquinas eran de un enorme coste.

Estos primeros sistemas operativos se limitaban a controlar y secuenciar la ejecución de programas y sus datos, que en aquella época estaban escritos en tarjetas perforadas. Las tarjetas perforadas llevaban escritas las instrucciones de programa en forma de agujeros en una cartulina, que una máquina lectora de tarjetas comunicaba al sistema operativo. Este iba



obteniendo las diferentes instrucciones en secuencia y controlando su ejecución.

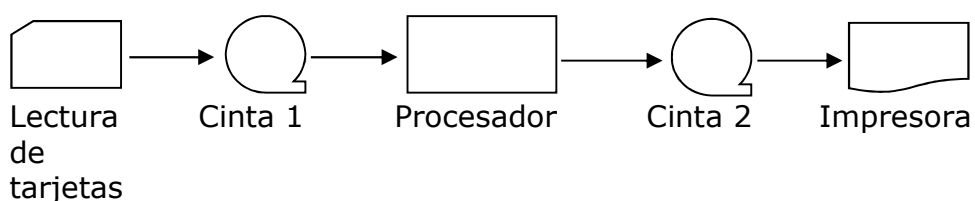


***Tarjeta perforada***

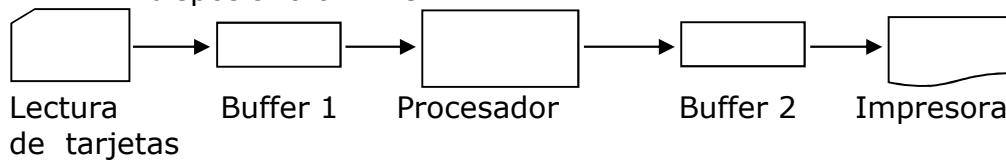
## 4.2 SEGUNDA GENERACION (1955-1965)

En esta época aparecen:

- Los transistores que hacen que las máquinas se reduzcan de tamaño.
- Los lenguajes de bajo nivel (assembler) y el lenguaje JCL.
- Los lenguajes de alto nivel que tenían que cargar un compilador que convertía el lenguaje en assembler de la máquina en la que se compilaba. Había que quitar el compilador de la cinta magnética e insertar el traductor, más tarde carga el programa objeto en memoria y lo ejecuta. Todos estos recursos aumentaban el número de pasos que había que dar.
- Más periféricos como las unidades de cinta magnética y una mejora notable de las impresoras. Una forma de reducir el tiempo de parada del procesador es pasar de los dispositivos más lentos a otros más rápidos con un sistema off-line. Si había una lectura de fichas se pasaba la información a una cinta y el procesador leía de ella. El procesador volcaba la información sobre otra cinta y después off-line pasaba de la cinta a la impresora. Este procesamiento, en general es más rápido, pero para un solo programa es más lento.



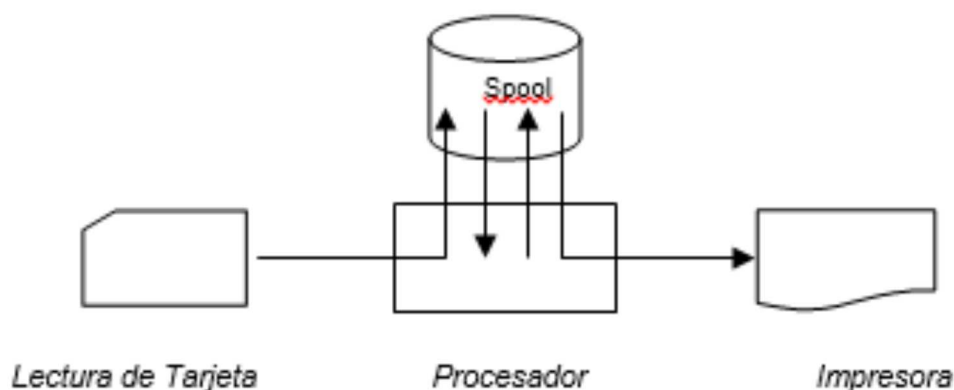
- Los buffer que son memorias intermedias donde pasa la información, el dispositivo leerá del buffer, eliminando así el dispositivo off-line.



### 4.3 TERCERA GENERACION (1965-1980)

En esta época aparece:

- La escalabilidad, que es la posibilidad de ampliar en un momento determinado las funciones de un sistema.
- La **multiprogramación**, es decir, que el procesador está trabajando continuamente. Consiste en dividir el espacio de memoria en varias partes, pudiendo tener un programa en cada una de esas partes. Cuando hay un proceso de E/S el S.O. comienza otro trabajo.
- Las unidades de disco. Ahora se utiliza el disco para realizar la función del buffer. Este disco se llama **spool** y sólo lo utilizaba el sistema operativo.



- Los **sistemas de tiempo compartido**, en donde el procesador reparte su tiempo entre los usuarios que están utilizándolo. Con el sistema de tiempo compartido el paso de los sistemas procesamiento por lotes es más reducido.
- Los **sistemas de tiempo real** que funcionan mediante una serie de sensores que tienen una respuesta inmediata. Son tiempos de respuesta inmediata.

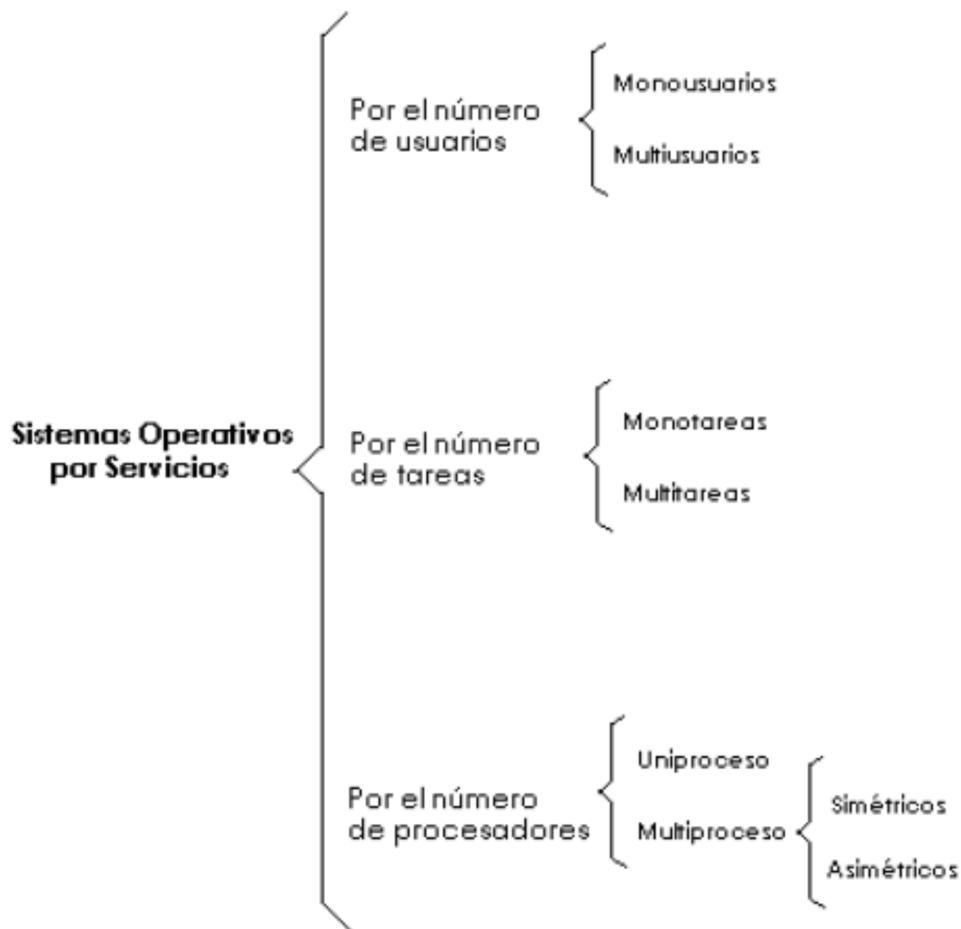
### 4.4 CUARTA GENERACION (1980 - hasta nuestros días)

En esta época aparece:

- **Conectividad**, que permite una gran libertad de comunicación pero hace que aparezcan problemas en la Seguridad, lo que impulsa el desarrollo de la criptografía que intenta asegurar la privacidad, la integridad del mensaje y la autenticación del mismo.
- El concepto de **Máquina Virtual** que simula otras máquinas en una plataforma concreta (Emuladores). Esto alcanza su mayor desarrollo con la plataforma Java que es un Lenguaje y una Máquina Virtual.
- Los **Sistemas de Gestión de Bases de datos**.

## 5. TIPOS DE SISTEMAS OPERATIVOS

Para ver los tipos de sistemas operativos hemos de tener en cuenta el **número de usuarios** que pueden utilizar el sistema. También tenemos que considerar los **procesos que dicho sistema pueda realizar a la vez**: uno o varios. Igualmente, dependerá del **número de procesadores con los que cuente el ordenador**. Por último tendremos en cuenta **el tiempo de respuesta del sistema**. A continuación vamos a clasificar a los sistemas operativos según esos puntos de vista.



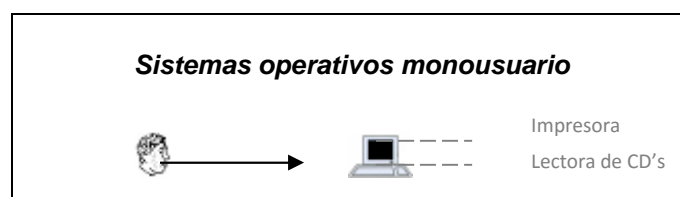
## 5.1 SEGÚN EL NÚMERO DE USUARIOS

Según el número de usuarios que pueden acceder a un ordenador y de acuerdo con el sistema operativo que les dé servicio, nos encontramos con los siguientes sistemas:

### 5.1.1 MONOUSUARIO

Sólo un usuario trabaja con un ordenador. En este sistema todos los dispositivos de hardware están a disposición de dicho usuario y no pueden ser utilizados por otros hasta que éste no finalice su sesión.

⇒ Sólo ese usuario puede utilizar los dispositivos hardware del ordenador: grabadora, impresora, ...



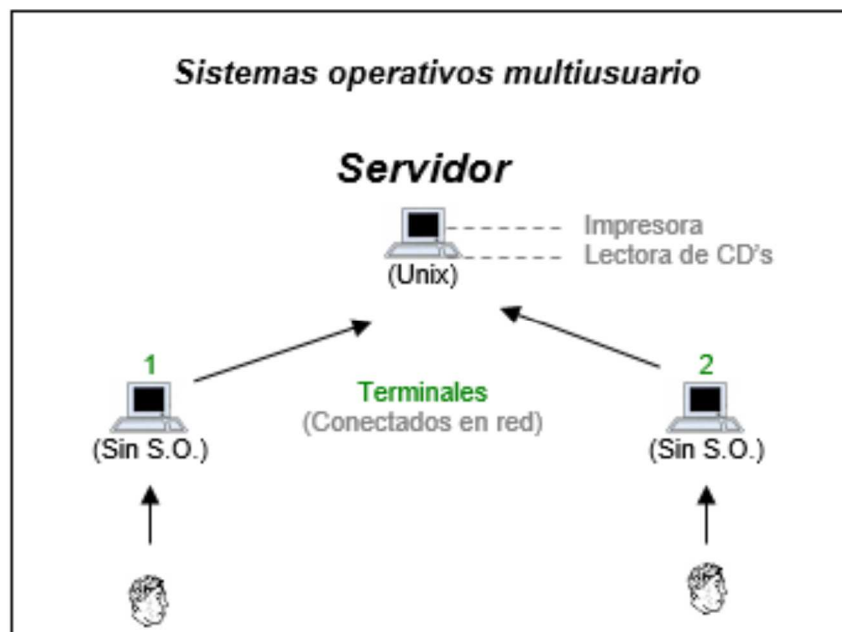
Ejemplos de sistemas operativos monousuario: MS-DOS, Windows 95, Windows 98, Windows ME  $\equiv$  Windows Millennium Edition, IBM-DOS, DR-DOS, PCOS, etc.

### 5.1.2 MUTIUSUARIO

Este tipo de sistemas se emplean especialmente en redes  $\Rightarrow$  Cuando aparece la palabra "Server" indica que es Multiusuario, pues en el servidor se crean cuentas para los diferentes usuarios.

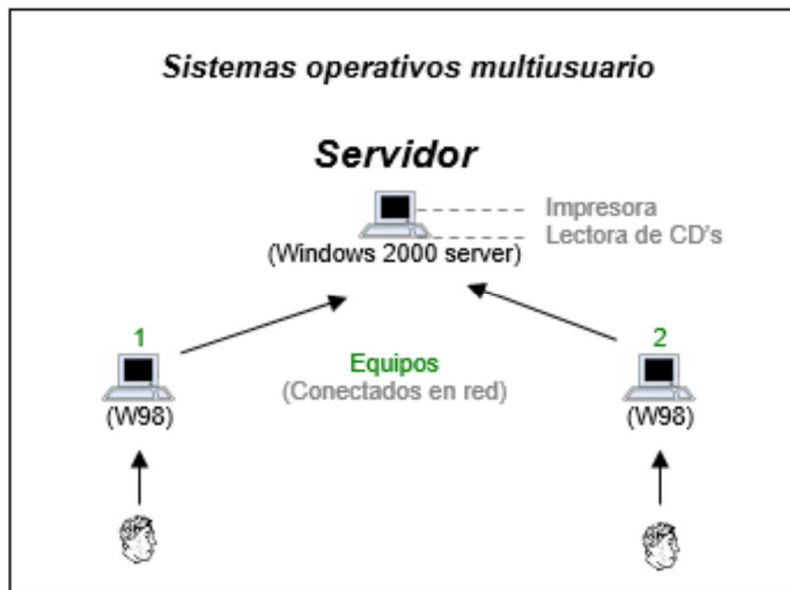
Varios usuarios pueden utilizar simultáneamente los recursos y dispositivos hardware del ordenador "Servidor", bien sea:

- Por medio de varias terminales conectadas al ordenador.



Los terminales 1 y 2 están compuestos de un monitor y un teclado el cual lleva incorporado una tarjeta de red para que los usuarios puedan conectarse al servidor.

- Mediante sesiones remotas en una red de comunicaciones.



En ambos casos Cada usuario tiene una cuenta en el servidor, por lo tanto pueden utilizar los recursos hardware del servidor.

Ejemplos de sistemas operativos multiusuario: UNIX, Novel, Windows NT Server, Windows 2000 Server, Windows 2000 Advanced Server (Es igual que el Windows 2000 Server pero con la posibilidad de conectar más terminales), VMS (Dígital), MVS (grandes equipos IBM), OS/400 (del IBM AS/400), windows XP (no puede ser servidor), windows 2000 Profesional (no puede ser servidor), etc.

Los recursos que se comparten son, normalmente, una combinación de:

- Procesador.
- Memoria.
- Almacenamiento secundario (en disco duro).
- Programas.
- Periféricos como impresoras, plóteres, escáneres, etc.

Para gestionar las impresoras, los trabajos enviados por varios usuarios se sitúan en colas de espera hasta que les llegue su turno.

En la Cola de espera tenemos:

- Trabajo1 de usuario1
- Trabajo2 de usuario1
- Trabajo1 de usuario2

⇒ Los trabajos se van imprimiendo según vayan llegando.

Estas colas se denominan **spool de impresión**.

## 5.2 SEGÚN EL NÚMERO DE TAREAS ( $\cong$ PROCESOS $\cong$ ventanas)

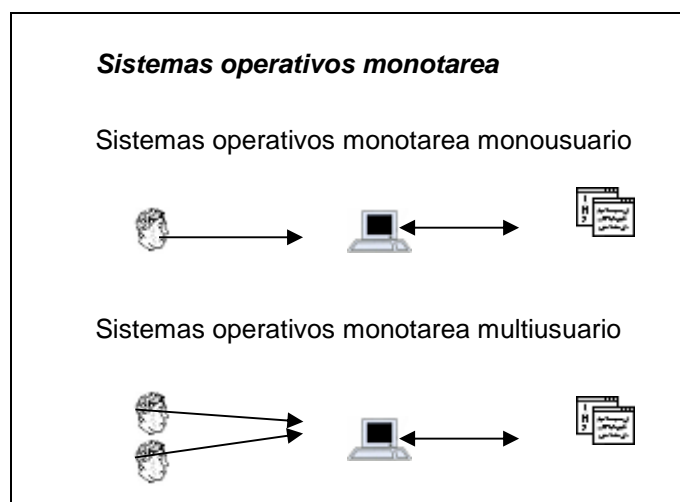
Esta clasificación se hace atendiendo al número de programas o procesos que puede realizar simultáneamente el ordenador o sistema informático.

### 5.2.1 MONOTAREA ( $\cong$ monoprogramación)

Los sistemas operativos monotareas son más primitivos solo pueden manejar un proceso en cada momento o que solo puede ejecutar las tareas de una en una. Por ejemplo, cuando el ordenador está imprimiendo un documento, no puede iniciar otro proceso ni responder a nuevas instrucciones hasta que se termine la impresión.

Esto no impide que el sistema pueda ser multiusuario; es decir, varios usuarios pueden intentar ejecutar sus programas en el mismo ordenador, pero de forma sucesiva. Para ello, se tienen que establecer las correspondientes colas o prioridades en la ejecución de los trabajos.

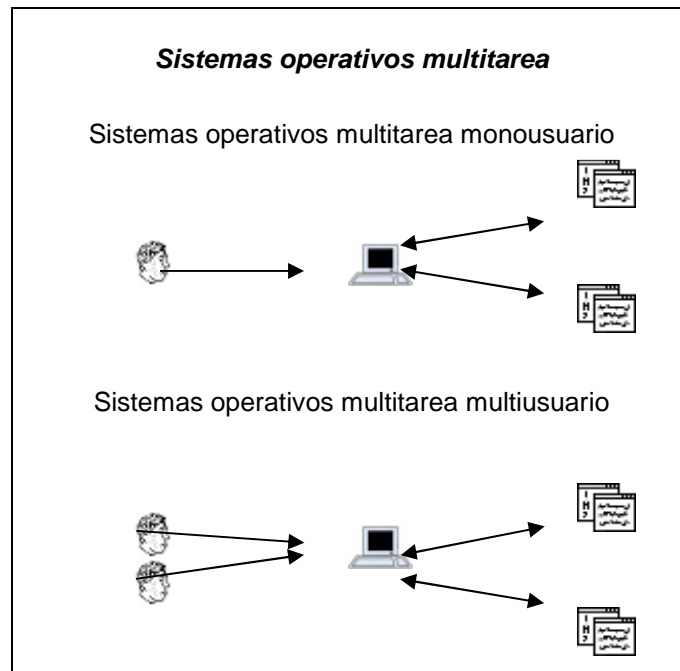
En este sistema, la atención del procesador estará dedicada a un solo programa hasta que finalice.



Ejemplo de sistema operativo monotarea: MS-DOS.

### 5.2.2 MULTITAREA ( $\cong$ multiprogramación)

El ordenador puede ejecutar varios programas o procesos a la vez. Para ello la UCP compartirá el tiempo de uso del procesador ( $\cong$  microprocesador) entre los diferentes programas que se van a ejecutar.



Ejemplo de sistema operativo multitarea: Windows NT Workstation, Windows NT Server, Windows 2000 Profesional, Windows 2000 Server, Windows 2000 Advanced Server, UNIX, Novell, etc. En general, la mayoría son sistemas multitarea.

### 5.3 SEGÚN EL NÚMERO DE PROCESADORES ( $\cong$ Microprocesadores).

Según que el ordenador cuente con uno o varios procesadores para realizar los procesos:

#### 5.3.1 MONOPROCESADOR ( $\cong$ MONOPROCESO)

El ordenador en el cual se utiliza el sistema operativo solo tiene un procesador y el sistema operativo solo es capaz de manejar un procesador. Todos los procesos del sistema pasaran por dicho procesador. Aunque el sistema pueda denominarse multitarea y multiusuario, y de hecho varias personas puedan trabajar con el mismo equipo, teniendo este un solo procesador, realmente los procesos que es capaz de ejecutar el procesador a la vez, es solo uno, aunque se reparte el tiempo de uso del procesador entre todos los procesos activos del sistema, simulando un multiproceso.

Ejemplos de sistemas operativos monoprocesador: MS-DOS, Windows 95, Windows 98, etc.



### 5.3.2 MULTIPROCESADOR ( $\cong$ MULTIPROCESO)

Si el sistema informático cuenta con dos o más procesadores, existen sistemas operativos capaces de gestionar varios procesadores a la vez, de esta forma se aprovecha mejor la capacidad del equipo en la ejecución de procesos entre varios procesadores, la utilización de los procesadores por parte del sistema puede ser de dos tipos:

- **Multiproceso Simétrico** (SMP, Symetrical MultiProcessing): En el cual el sistema operativo utiliza los procesadores por igual alternando el uso de los mismos de forma simultánea.
- **Multiproceso Asimétrico** (AMP, Asymetrical MultiProcessing): El sistema reparte las tareas que están realizando los procesadores, determinando que procesos ejecuta cada procesador.

### 5.4 SEGÚN EL TIEMPO DE RESPUESTA

Esta clasificación se hace teniendo en cuenta el tiempo que tarda el sistema en obtener los resultados después de lanzar un programa en ejecución:

#### 5.4.1 PROCESOS POR LOTES (BATCH)

Los sistemas operativos por lotes, procesan una gran cantidad de trabajos con poca o ninguna interacción entre los usuarios y los programas en ejecución. Cuando estos sistemas son bien planeados, pueden tener un tiempo de ejecución muy alto, porque el procesador es mejor utilizado y los sistemas operativos pueden ser simples, debido a la secuencialidad de la ejecución de los trabajos.

Algunas características de los sistemas operativos por lotes son las siguientes:

- Requiere que el programa, datos y órdenes al sistema sean remitidos todos juntos en forma de lote.
- Permiten poca o ninguna interacción con el usuario.
- Mayor potencial de utilización de recursos que procesamiento serial simple en sistemas multiusuarios.
- Conveniente para programas de largos tiempos de ejecución.

Ejemplos de sistemas operativos por lotes: Todos.

#### 5.4.2 TIEMPO REAL ( $\cong$ REAL TIME)

Los sistemas operativos de tiempo real son aquellos en donde no tiene importancia el usuario, sino los procesos. Se utilizan en entornos donde son procesados un gran número de sucesos o eventos.

Son contruidos para aplicaciones muy específicas, tales como: tráfico aéreo, bolsas de valores, etc.

Algunos campos de aplicación son los siguientes:

- Control de trenes.
- Telecomunicaciones.
- Sistemas de fabricación integrada.
- Control de edificios, etc.

Algunas características de los sistemas operativos de tiempo real son:

- Su objetivo es proporcionar rápidos tiempos de respuesta.
- Procesa ráfagas de miles de interrupciones por segundo sin perder algún proceso.
- Poco movimiento de programas entre almacenamiento secundario y memoria.
- Los procesos de mayor prioridad expropia recursos a otros procesos.

Ejemplos de sistemas operativos por lotes: Todos.

### 5.4.3 TIEMPO COMPARTIDO

Comparte el tiempo del microprocesador entre los procesos lanzados. Cada proceso utilizará fracciones de tiempo de ejecución de la UCP hasta que finalice. En este caso, parece que el usuario dedica la UCP exclusivamente para él; pero esto no es cierto, ya que, aunque el usuario no lo perciba, la UCP está dedicada a varios procesos a la vez.

Ejemplos de sistemas operativos en tiempo compartido: Todos los actuales.

En el caso de tener 2 microprocesadores (s.o. multiprocesador) – Si lanzamos 3 tareas seguirá compartiendo el microprocesador. Por lo tanto aunque haya más de un microprocesador, en algún momento, va a tener que trabajar en tiempo compartido.

## 5.5 SEGÚN SU DISPONIBILIDAD

Dentro de esta clasificación tendremos:

### 5.5.1 PROPIETARIOS

Los sistemas operativos propietarios son aquellos que son propiedad de alguna empresa.

Características:

- Se necesitan licencias de uso.
- No se dispone de acceso a su código fuente para modificarlo ni distribuirlo. Ej - Windows.

### 5.5.2 LIBRES

Los sistemas operativos libres son aquellos que garantizan las libertades de:

- Usar el programa.
- Estudiar cómo funciona el programa y modificarlo, adaptándolo a las necesidades que tuviera el usuario.
- Distribuir copias del programa, con lo que se puede ayudar a otros usuarios.
- Mejorar el programa y hacer públicas dichas mejoras, de modo que todos los usuarios se beneficien de ello.

Sistemas operativos						
Sistemas/familias	Creador	Licencia	Usuarios	Tareas	Procesadores	interfaz
MS-DOS, DR-DOS, PC-DOS (abandonados)	Microsoft, Digital, Resecach, IBM	Propietario	Monousuario	monotarea	Monoprocesador	Texto
Windows 98, 98, ME, NT, 2000 (abandonados o en desuso)	Microsoft	Propietario	Multiusuario (usuarios no concurrentes)	multitarea	Monoprocesador	Gráfica
Windows XP, Vista	Microsoft	Propietario	Multiusuario	Multitarea	Multiprocesador	Gráfica
Windows 2000, 2003 y 2008	Microsoft	Propietario	Multiusuario	Multitarea	Multiprocesador	Gráfica
Mac OS 7, 8, 9, X	Apple	Propietario	Multiusuario	Multitarea	Multiprocesador	Gráfica
UNIX	Open Group	Propietario	Multiusuario	Multitarea	Multiprocesador	Gráfica
Linux (basado en UNIX)	Depende de distribución	Propietario	Multiusuario	Multitarea	Multiprocesador	Ambas

## 6. TIPOS DE APLICACIONES ( $\cong$ SOFTWARE)

Existen los siguientes tipos de software:

### 6.1 GRATUITO ( $\cong$ freeware) O COMERCIAL

Hace referencia al pago.

#### 6.1.1 SOFTWARE GRATUITO

El software gratuito, como su nombre indica, se puede adquirir gratis o con alguna limitación. Esas limitaciones pueden ser:

- Al adquirir la aplicación, permite utilizarla pero le falta alguna opción como: Guardar, Imprimir,...  $\Rightarrow$  Crearía una dependencia para que el usuario termine comprando dicha aplicación.
- Límite de tiempo para probar la aplicación, por ejemplo 30 días.

Ejemplos de software gratuito:

- Ubuntu - Sistema operativo, es una variante de Linux.
- Mozilla Firefox - Navegador web.
- OpenOffice - Paquete ofimático.

### 6.1.2 SOFTWARE COMERCIAL

El software comercial es aquel que hay que comprar para poder utilizarlo.

Ejemplos de software comercial:

- Windows XP, Windows vista, Windows 7,... - Sistemas operativos.
- Internet Explorer - Navegador web.
- Microsoft Office - Paquete ofimático.
- Cualquier aplicación de Microsoft.

## 6.2 LIBRE O PROPIETARIO

Hace referencia al código.

### 6.2.1 SOFTWARE LIBRE ( $\cong$ free software (no confundir con software gratis) $\cong$ software de código abierto)

El software libre se basa en la distribución del código fuente junto con el programa. Puede ser usado, copiado, estudiado, modificado y redistribuido libremente ( $\cong$  4 libertades). Es decir, se puede acceder al código del programa para modificarlo.

El software libre suele estar disponible gratuitamente o al precio de coste de la distribución  $\Rightarrow$  software libre  $\neq$  software gratuito.

Ejemplos de software libre:

- El OpenOffice - Se puede acceder al código de la aplicación OpenOffice, modificarlo según las necesidades del usuario, y cambiarle el nombre pasándose a llamar, por ejemplo, OpenOfficeNew

### 6.2.2 SOFTWARE PROPIETARIO ( $\cong$ software no libre $\cong$ software de código cerrado)

Es cualquier programa informático en el que el usuario tiene limitaciones para usarlo, modificarlo o redistribuirlo (con o sin modificaciones). Es decir, el código del programa no se puede modificar.

Ejemplos de software Propietario:

- Microsoft Office - No se puede acceder al código de la aplicación Microsoft Office, para modificarlo según las necesidades del usuario.

## 6.3 OPENSOURCE O PRIVATIVO

### 6.3.1 SOFTWARE OPENSOURCE

Dejan el código abierto al usuario.

### 6.3.2 SOFTWARE PRIVATIVO

Su código fuente no está disponible o el acceso a él se encuentra restringido.

## 7. TIPOS DE LICENCIAS

Existen los siguientes tipos de licencias:

### 7.1 OEM

Son licencias de software que son adquiridas en la compra de un ordenador con software legalmente preinstalado. Cualquier ordenador nuevo con Microsoft Windows o Microsoft Office ya instalado, tiene licencias OEM.

Los programas adquiridos bajo este tipo de licencia no se pueden vender ni ceder a terceros.

Ejemplo de licencias OEM: Windows 2008 y Windows 2010.

### 7.2 RETAIL

Estas licencias se pueden comprar, por lo tanto el programa es de la entera propiedad del usuario, pudiendo éste cederlo libremente a terceros o venderlo.

Ejemplo de licencias RETAIL: Windows 2008 y Windows 2010.

### 7.3 LICENCIAS POR VOLUMEN

Es un tipo de licencia de software destinado a grandes usuarios (empresas), normalmente bajo unas condiciones similares a las de las licencias OEM, aunque sin estar supeditadas a equipos nuevos.

Estas licencias se venden en paquetes de x número de licencias, por ejemplo en paquetes de 25 licencias como mínimo.

Este tipo de licencias no se puede ceder a terceros ni total ni parcialmente.

## 8. GESTORES DE ARRANQUE

En caso de que haya instalados varios sistemas operativos en un mismo ordenador, hay que utilizar un sistema para poder seleccionar qué sistema operativo se desea iniciar.

El gestor de arranque es un pequeño programa que se ejecuta una vez completado el inicio normal de la BIOS y que permite seleccionar el sistema operativo en caso de arranque múltiple.

Existen los siguientes gestores de arranque:

### 8.1 NTLDR ( $\cong$ NT Loader)

**NTLDR** (abreviatura de NT Loader) es el archivo encargado del arranque del Sistema Operativo en las primeras versiones de Microsoft Windows NT, incluyendo Windows XP y Windows Server 2003 (A partir de las versiones superiores como Windows Vista y Windows 7 se usa BOOTMGR). El NTLDR se encuentra usualmente en el disco duro principal, pero también puede encontrarse en dispositivos portátiles como CD-ROM, memorias USB, o disquetes.

NTLDR requiere, como mínimo, que dos archivos se encuentren en el directorio raíz del volumen de inicio:

- **NTLDR**, que se encarga de cargar el sistema operativo.
- **boot.ini**

Su archivo de configuración es Boot.ini, un sencillo archivo de texto que se puede ver en la raíz de la unidad principal.

En él se relacionan los sistemas operativos instalados en el equipo, cuál de ellos es el predeterminado, el timeout o tiempo de espera, etc. Boot.ini se puede modificar usando el Bloc de notas u otro editor de texto.

Si el archivo NTLDR no se encuentra en el disco, la computadora enviará un mensaje de error informándolo.

En Windows Vista y posteriores (Windows Server 2008, Windows 7 y Windows 8), el NTLDR fue reemplazado por dos componentes llamados winload.exe y Windows Boot Manager (BOOTMGR este se puede instalar a los discos duro y otros dispositivo con programas como bootice).

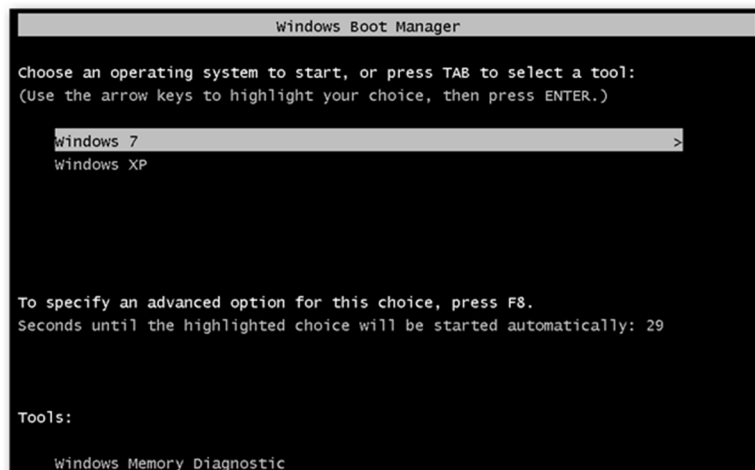
## 8.2 BOOT MANAGER ( $\cong$ BOOTMGR $\cong$ Administrador de arranque de Windows)

**Boot manager** es el **gestor de arranque propio de Windows**. Cada PC que tiene instalado alguno de los SO de la familia de Windows, tiene instalado Boot manager. Este gestor solo permite el arranque de sistemas operativos propios de Windows, por lo que, si se desea instalar otro SO que no sea de la familia de Windows, es recomendable instalar otro gestor de arranque.

Este gestor se introduce con Windows Vista y se usa también en Windows 7 y superiores, el nuevo Administrador de arranque de Windows (Windows Boot Manager) **BCD** (Boot Configuration Data), en inglés: Datos de la Configuración de arranque.

La información se guarda en una base de datos similar al Registro de Windows, en un archivo llamado: bootmgr y en C:\Boot\BCD.LOG.

Para modificarla se usa el comando BCDEDIT.



Controla el proceso de arranque mostrando el menú multiarranque (si hubiera más de un sistema operativo instalado en el disco).

Después llamará a los siguientes archivos:

- **WinLoad.exe** - Es el cargador del sistema operativos Windows.
- **ntoskrnl.exe** - Se encarga del resto del arranque del sistema.

## 8.3 LILO ( $\cong$ Linux Loader)

Es un **gestor de arranque de Linux** que permite iniciar este sistema operativo junto con otras plataformas (como Windows) que haya en el mismo ordenador. Funciona en una variedad de sistemas de archivos y puede arrancar un sistema operativo desde el disco duro o desde un disco flexible externo.

## 8.4 GRUB

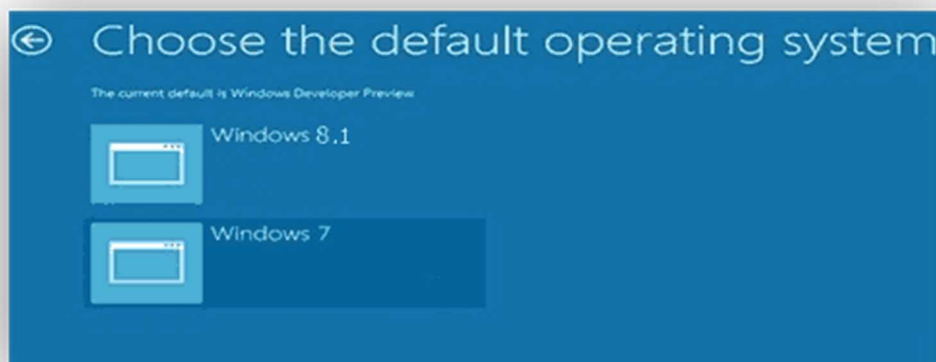
Es un gestor de arranque más moderno y flexible que LILO, ya que permite que el administrador ejecute cualquier comando desde la línea de comando de GRUB. Entre todas sus características hay que destacar la posibilidad de incluir múltiples formatos de ejecutables, el arranque de sistemas operativos no-multiarranque, una agradable interfaz de usuario y una interfaz de línea de comando muy flexible.

## 8.5 GESTORES DE ARRANQUE DE WINDOWS

Windows por lo general instala uno de los gestores de arranque en nuestro sistema, sin embargo, cuando únicamente tenemos un sistema operativo instalado este gestor no aparece en el arranque cargando automáticamente el sistema.

Si vamos a instalar varios sistemas operativos Windows (por ejemplo Windows 7 y Windows 8) podemos aprovechar perfectamente uno de los gestores de arranque de Windows para poder elegir el sistema operativo que queremos arrancar por defecto en el arranque del sistema. Su instalación y configuración es muy sencilla, lo único que debemos hacer es asegurarnos de instalar en primer lugar el sistema operativo más antiguo (Windows 7 en este caso) y a continuación el más reciente (Windows 8) para que haya total compatibilidad entre ellos.

Al instalar Windows 8 el proceso de instalación detectará una partición previa con Windows 7 y automáticamente añadirá una entrada al gestor de arranque de manera que al arrancar el sistema podremos elegir el que queremos cargar y utilizar.



## 8.6 GRUB Y LILO, LOS GESTORES DE ARRANQUE DE LINUX

Aunque el gestor de arranque de Microsoft funciona con otros sistemas operativos ha sido diseñado para funcionar con otros Windows, pudiendo llegar a dar error al añadir un sistema Linux o un Hackintosh. Linux instala



también uno de sus propios gestores de arranque que son mucho más completos y compatibles que los gestores de arranque de Windows.

Existen varios gestores de arranque que se instalan según nuestra distribución, sin embargo, uno de los más potentes es Grub. Grub nos va a permitir seleccionar fácilmente el sistema operativo que queremos arrancar en una lista sencilla compatible con todo tipo de sistemas (Windows, Linux, Hackintosh, etc).

Para instalar y configurar Grub lo primero que debemos hacer es instalar todos los sistemas operativos que queramos tener en sus correspondientes particiones. Una vez instalados los sistemas el último que instalaremos será nuestro Linux (concretamente Ubuntu) para que durante el proceso de instalación se instale Grub.

Grub detectará automáticamente todos los sistemas operativos instalados en todas las particiones del equipo y añadirá una entrada para cada uno de ellos sin que nosotros tengamos que hacer nada. Si eliminamos un sistema o instalamos uno nuevo simplemente debemos iniciar nuestro Linux y teclear "sudo update-grub" para actualizar la configuración con los nuevos sistemas.

Grub, aparte de ser una herramienta muy completa y completamente funcional también es totalmente personalizable pudiendo obtener apariencias similares a la siguiente.



## 8.7 GESTORES DE ARRANQUE INDEPENDIENTES

A continuación vamos a ver dos ejemplos de gestores de arranque independientes al sistema operativo que estemos utilizando.

### 8.7.1 GESTOR DE ARRANQUE BOOTIT

**BootIt** es un gestor de arranque independiente al sistema operativo que nos va a permitir configurar un arranque dual con los sistemas operativos que tengamos instalados. Esta aplicación nos va a permitir configurar los discos (particionar, ordenador, formatear, etc) y crear un completo gestor de arranque mucho más completo que las opciones

anteriores (podemos, por ejemplo, proteger un sistema operativo con contraseña para evitar el acceso a él o utilizar un formato de archivos propio para crear más de 4 particiones primarias en un disco).

BootIt es un completo gestor y administrador de discos imprescindible para los usuarios más avanzados y más exigentes, sin embargo, es una aplicación de pago que puede no estar al alcance de algunos usuarios y que frente a potentes alternativas libres como Grub hace que la inversión no merezca la pena.



### 8.7.2 GAG, EL GESTOR DE ARRANQUE GRAFICO

GAG es un programa gestor de arranque independiente al sistema operativo que nos va a permitir configurar un arranque dual con los sistemas operativos que tengamos instalados.

Sus características más importantes son:

- Permite arrancar hasta 9 sistemas operativos diferentes.
- Puede arrancar sistemas operativos instalados tanto en particiones primarias como extendidas, en cualquiera de los discos duros instalados en el ordenador.
- Puede ser instalado desde casi cualquier sistema operativo.
- No necesita una partición propia, sino que se instala en la primera pista del disco duro, la cual se encuentra, por diseño, reservada para este tipo de funciones. También puede ser instalado en disquete, sin tocar para nada el disco duro.
- Incluye un temporizador que permite arrancar un sistema operativo por defecto.
- La configuración del programa puede ser protegida con clave.
- Todo el programa funciona en modo gráfico (necesita una VGA o superior para funcionar), e incluye multitud de iconos para cada tipo de sistema operativo disponible en PC.
- Oculta particiones primarias de modo que se pueden tener varios DOS y/o Windows en un mismo disco duro.
- Permite poner claves independientes a cada sistema operativo, para restringir el acceso a cada uno.

- Fácil de traducir a cualquier lengua.
- Puede intercambiar discos duros, permitiendo arrancar desde el segundo disco duro sistemas operativos como MS-DOS.
- Incluye el sistema SafeBoot, que permite seguir arrancando el disco duro incluso en caso de que GAG sea sobrescrito por accidente.
- Soporta varios tipos de teclados internacionales (QWERTY, AZERTY, QWERTZ y DVORAK).
- Soporta discos duros de hasta 4 terabytes (4096 gigabytes).
- Totalmente gratuito (se distribuye bajo licencia GPL con fuentes incluidas).

