Cuerpo de Técnicos Auxiliares de Informática de la Administración del Estado

opositaonline.com

"Elige tu destino · Cambia tu vida · Sé feliz"

II. Tecnología básica

TEMA 04
SISTEMAS OPERATIVOS

Contenidos:

- Características y elementos constitutivos.
- Sistemas Windows.
- Sistemas Unix y Linux.
- Sistemas operativos para dispositivos móviles.

TEMA II - 04.- SISTEMAS OPERATIVOS

CONTENIDOS

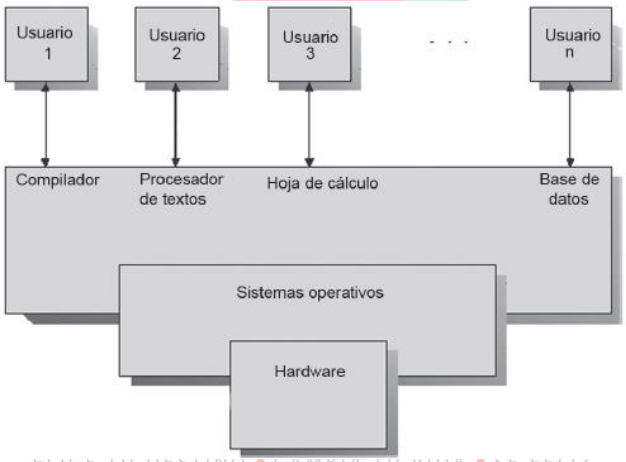
"El éxito no se logra solo con cualidades especiales. Es sobre todo, un trabajo de constancia, de método y de organización" - J.P. Sergent (Pintor)

1. Estructuras fundamentales de datos. Organizaciones de ficheros

Un **sistema operativo** es un **programa o conjunto de programas** que actúa como **intermediario entre el usuario y el hardware del ordenador**, **gestionando** los **recursos** del **sistema y optimizando** su uso.

El sistema operativo **es en sí mismo un programa**, pero un programa muy especial y quizás el más complejo e importante. **Cuando se conecta un ordenador se carga parte del sistema operativo en** la **memoria y se ejecuta**. El sistema operativo despierta al ordenador y hace que reconozca a la CPU, la memoria, las unidades de disco y cualquier otro dispositivo conectado a ella, como el teclado, el ratón, la impresora, etc., verificando así que no existan errores de conexión y que todos los dispositivos se han reconocido y trabajan correctamente.

El sistema operativo presenta al usuario la máquina, de una forma más fácil de manejar y programar, que el hardware que está por debajo, es decir, un usuario normal, simplemente abre los ficheros que grabó en un disco, sin preocupase por la disposición de los bits en el medio físico, los tiempos de espera del motor del disco, la posición de un cabezal, el acceso de otros usuarios, etc.



El S.O. como intermediario entre los usuarios y el HW

1.1. Funciones del Sistema Operativo

A continuación, se muestran las **funciones principales** que realiza todo sistema operativo:

- **Control de la ejecución de los programas**: para ello, **acepta** los **trabajos**, administra la manera en que se realizan, les **asigna** los **recursos** y los conserva hasta su finalización.
- **Administración de periféricos**: **coordinando y manipulando** los **dispositivos** conectados al ordenador.
- **Gestión de permisos y de usuarios**: adjudica los **permisos de acceso** a los usuarios y evita que las acciones de uno afecten el trabajo que está realizando otro.
- **Control de concurrencia**: establece **prioridades** cuando diferentes procesos solicitan el mismo recurso.
- **Control de errores**: gestiona los errores de **hardware y** la **pérdida de datos**.
- **Administración de memoria**: **asigna memoria a los procesos** y gestiona su uso.

- **Control de seguridad**: debe proporcionar **seguridad** tanto **para** los **usuarios como para el software** y la información almacenada en los sistemas.

En concordancia con estas funciones principales, **es posible analizar la estructura de un sistema operativo en cinco niveles**. Los **primeros dos** niveles entrarían dentro de la parte del **sistema operativo dependiente del hardware**, el **resto** de los niveles pertenecen a la **parte portable** del mismo.

Cada uno de los **niveles se comunica con el inmediatamente inferior y superior** coordinando sus funciones.

- **Nivel 1 - Gestión del procesador**: en este nivel se encuentra la parte del sistema operativo encargada de la **gestión de la CPU**.
- **Nivel 2 - Gestión de memoria**: este nivel es el **encargado de repartir la memoria** disponible **entre los procesos**. Se realizan funciones de asignación y liberación de memoria, y el control de violación de acceso a zonas de memoria no permitidas.
- **Nivel 3 - Gestión de procesos**: este nivel es el encargado de la **creación y destrucción de los procesos**, intercambio de mensajes y detección y arranque de los mismos.

- **Nivel 4 - Gestión de dispositivos**: en este nivel se realiza la **gestión de las entradas/salidas (E/S)** en función de los dispositivos existentes. Entre otras, se encarga de las funciones de creación de procesos de E/S, asignación y liberación de dispositivos E/S, y planificación de la E/S.
- **Nivel 5 - Gestión de la información**: el objetivo de este nivel es el de **gestionar el espacio de nombres lógicos**, utilizados para simplificar el acceso a los recursos, ya que mediante estos se sustituyen rutas de acceso que pueden ser muy largas y difíciles de recordar por un solo nombre, encargándose el sistema operativo, de forma totalmente transparente para el usuario, de realizar esa búsqueda de ruta. Otro de sus cometidos es la **protección de la información** realizando funciones de creación y destrucción de ficheros y directorios, apertura y cierre de ficheros, lectura y escritura de ficheros y protección de acceso.

1.2. Clasificación de los sistemas operativos

En este apartado se describirán las características que clasifican a los sistemas operativos, básicamente **se cubrirán tres clasificaciones**:

1. Sistemas operativos **por su estructura** (**visión interna**),
2. Sistemas operativos **por los servicios que ofrecen** y, finalmente,
3. Sistemas operativos **por la forma en que ofrecen sus servicios** (**visión externa**).

Sistemas Operativos por su estructura

Se deben observar dos tipos de requisitos cuando se construye un sistema operativo, los cuales son:

- **Requisitos de usuario**: sistema fácil de usar y de aprender, seguro, rápido y adecuado al uso al que se le quiere destinar.

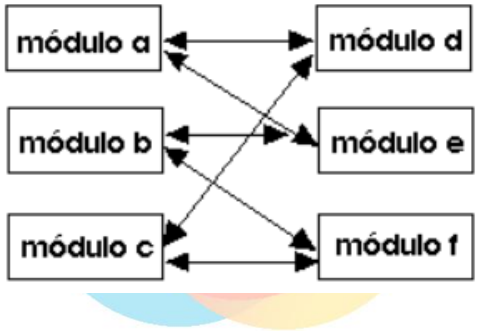
- **Requisitos del software**: donde se engloban aspectos como el mantenimiento, forma de operación, restricciones de uso, eficiencia, tolerancia frente a los errores y flexibilidad.

A continuación, se describen las distintas estructuras que presentan los actuales sistemas operativos para satisfacer las necesidades que de ellos se quieren obtener.

ESTRUCTURA MONOLÍTICA

Es la **estructura de los primeros sistemas operativos constituidos** fundamentalmente **por un solo programa compuesto de un conjunto de rutinas entrelazadas** de tal forma que **cada una puede llamar a cualquier otra** (ver figura siguiente). Las **características** fundamentales de este tipo de estructura son:

- **Construcción del programa final a base de módulos compilados** separadamente que **se unen a través del compilador**.
- **Buena definición de parámetros de enlace entre las distintas rutinas existentes**, que puede provocar mucho acoplamiento.
- **Carecen de protecciones y privilegios al entrar a rutinas que manejan diferentes aspectos de los recursos de la computadora**, como memoria, disco, etc.



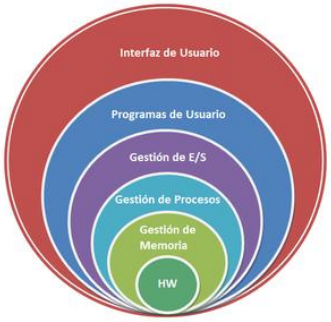
Generalmente están hechos a medida, por lo que **son eficientes y rápidos en su ejecución** y gestión, **pero** por lo mismo **carecen de flexibilidad** para soportar diferentes ambientes de trabajo o tipos de aplicaciones.

ESTRUCTURA JERÁRQUICA

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software, del sistema operativo, donde una parte del sistema contiene "subpartes" y se organizan en forma de niveles.

Capa 5 - Usuario
Capa 4 - Archivos
Capa 3 - Entrada/Salida
Capa 2 - Comunicaciones
Capa 1 - Memoria
Capa 0 - Gestión CPU
Capa 1 - Hardware

Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interface con el resto de elementos.



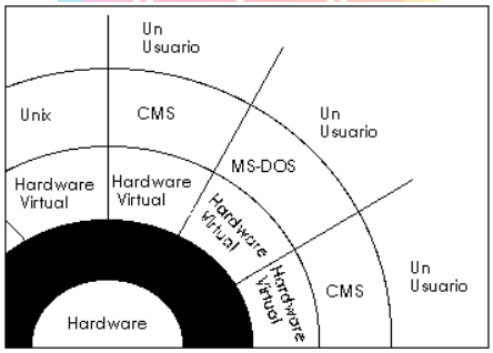
Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero de los cuales fue denominado THE (*Technische Hogeschool, Eindhoven*), de Dijkstra, que se utilizó con fines didácticos (ver figura). Se puede pensar también en estos sistemas como si fueran `multicapa'. Multics y Unix pertenecen a esta categoría.

En esta estructura se basan prácticamente la mayoría de los sistemas operativos actuales. **Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos**.

MÁQUINA VIRTUAL

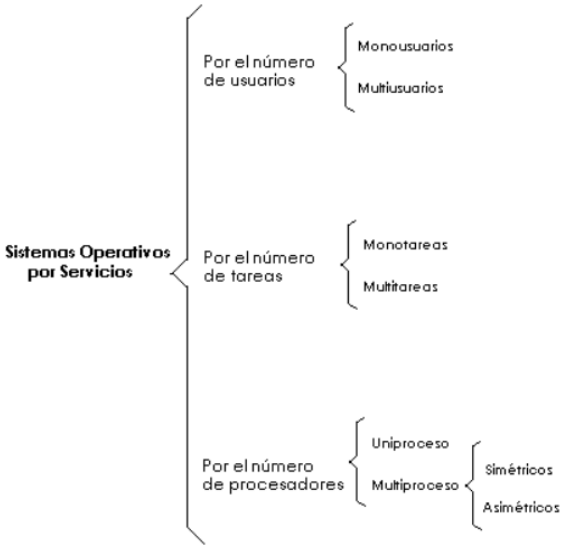
Se trata de un **tipo de sistemas operativos que presentan una interface a cada proceso, mostrando una máquina que parece idéntica a la máquina real subyacente**. Estos sistemas operativos separan dos conceptos que suelen estar unidos en el resto de sistemas: la multiprogramación y la máquina extendida. El **objetivo de los sistemas operativos de máquina virtual** es el de **integrar distintos sistemas operativos dando la sensación de ser varias máquinas diferentes**.

El **núcleo de estos sistemas operativos** se denomina **monitor virtual** y tiene **como misión llevar a cabo la multiprogramación**, presentando a los niveles superiores tantas máquinas virtuales como se soliciten. Estas máquinas virtuales no son máquinas extendidas, sino una réplica de la máquina real, de manera que en cada una de ellas se pueda ejecutar un sistema operativo diferente, que será el que ofrezca la máquina extendida al usuario (ver figura siguiente).



Sistemas Operativos por servicios

Esta **clasificación es la más usada habitualmente y la más conocida desde el punto de vista del usuario final.**



MONOUSUARIOS

Los sistemas operativos monousuarios **son aquellos que soportan a un solo usuario a la vez, sin importar** el **número de procesadores** que tenga la computadora o el **número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo**.

MULTIUSUARIOS

Los sistemas operativos multiusuarios **son capaces de dar servicio a más de un usuario a la vez**, ya sea por medio de **varias terminales conectadas** a la computadora o por medio de **sesiones remotas** en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario puede ejecutar simultáneamente.

MONOTAREAS

Los sistemas monotarea **son aquellos que sólo permiten una tarea a la vez por usuario**. **Puede darse el caso de un sistema multiusuario y monotarea**, en el cual **se admiten varios usuarios al mismo tiempo, pero cada uno de ellos** puede estar haciendo **solo una tarea a la vez**.

MULTITAREAS

Un sistema operativo multitarea es aquel que **permite al usuario estar realizando varias tareas al mismo tiempo**. Por ejemplo, puede estar editando el código fuente de un programa durante su depuración mientras compila otro programa, a la vez que está recibiendo correo electrónico en un proceso en background. Es común encontrar en ellos interfaces gráficas orientadas al uso de menús y el ratón, lo cual permite un rápido intercambio entre las tareas para el usuario, mejorando su productividad.

UNIPROCESO

Un sistema operativo uniproceso es **aquel que es capaz de manejar solamente un procesador de la computadora**, de manera que si la computadora tuviese más de uno le sería inútil.

MULTIPROCESO

Un sistema operativo multiproceso **se refiere al número de procesadores del sistema, que es más de uno y éste es capaz de usarlos todos para distribuir su carga de trabajo**. **Generalmente** estos sistemas **trabajan de dos formas: simétrica o asimétricamente:**

- Cuando se trabaja **de manera asimétrica**, el **sistema operativo selecciona a uno de los procesadores el cual jugará el papel de procesador maestro** y servirá como **pivote para distribuir la carga a los demás procesadores**, que reciben el nombre de **esclavos**. El **maestro ejecuta el código del S.O.** y el **resto ejecutan trabajos de usuario**.
- Cuando se trabaja **de manera simétrica**, los **procesos o partes de ellos** (*threads*) son **enviados indistintamente a cualquiera de los procesadores disponibles**, teniendo, teóricamente, una **mejor distribución y equilibrio en la carga de trabajo** bajo este esquema.

Sistemas Operativos por la forma de ofrecer sus servicios

Esta clasificación también **se refiere a una visión externa**, que en este caso se refiere a la del usuario, el **cómo accede a los servicios**. Bajo esta clasificación se pueden detectar **dos tipos principales: sistemas operativos de red y sistemas operativos distribuidos**.

SISTEMAS OPERATIVOS DE RED

Los sistemas operativos de red se definen como aq**uellos que tienen la capacidad de interactuar con sistemas operativos en otras computadoras** por medio de un **medio de transmisión** con el objeto de **intercambiar información, transferir archivos, ejecutar comandos remotos…** El punto crucial de estos sistemas es que **el usuario debe saber la sintaxis de un conjunto de comandos o llamadas al sistema para ejecutar estas operaciones**, además de la ubicación de los recursos que desee acceder.

SISTEMAS OPERATIVOS DISTRIBUIDOS

Los sistemas operativos distribuidos **abarcan los servicios de los de red, logrando integrar recursos** (impresoras, unidades de respaldo, memoria, procesos, unidades centrales de proceso) **en una sola máquina virtual que el usuario accede en forma transparente**. Es decir, ahora **el usuario ya no necesita saber la ubicación de los recursos**, sino que los conoce por nombre y **simplemente los usa como si todos ellos fuesen locales a su lugar de trabajo habitual**.

1.3. Procesos

Dentro de las **operaciones más básicas y, a la vez, más complejas** de nuestro ordenador encontramos los **procesos**. Siempre que le pidamos a nuestro equipo que haga algo, los procesos asumirán el trabajo y de esta manera el microprocesador ejecutará el plan que indique el sistema operativo mediante los procesos.

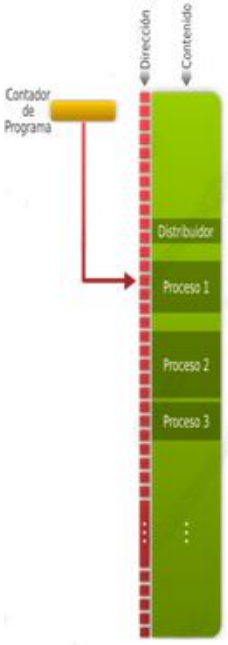
Desde el **punto de vista del sistema operativo**, un **proceso** es la **entidad mínima individualmente planificable, consta de código** (instrucciones máquina y llamadas al S.O.) **y datos y se caracteriza por sus atributos** (prioridad, permisos de acceso, …) **y estado dinámico**.

Un **proceso** es un concepto manejado por el sistema operativo que consiste en el **conjunto formado por**:

- **Las instrucciones de un programa** destinadas a ser ejecutadas por el microprocesador
- **Su estado de ejecución** en un momento dado, esto es, los valores de los registros de la CPU para dicho programa.
- **Su memoria de trabajo**, es decir, la memoria que ha reservado y sus contenidos.
- **Otra información** que permite al sistema operativo su **planificación**.

Esta definición varía ligeramente en el caso de **sistemas operativos multihilo**, donde un **proceso consta de uno o más hilos**, la **memoria de trabajo** (compartida por todos los hilos) **y** la **información de planificación**. **Cada hilo** consta de **instrucciones y estado de ejecución**.

Los **procesos son creados y destruidos por el sistema operativo**, que también **se debe hacer cargo de la comunicación entre procesos**, pero esto lo hace a petición de otros procesos. El **mecanismo** por el cual un **proceso crea otro proceso se denomina bifurcación** (*fork*). Los nuevos procesos son independientes y no comparten memoria (es decir, información) con el proceso que los ha creado.



Un **proceso también se define como un programa en ejecución**. Cada proceso se compone de un código que se ejecuta y una estructura de datos, estando ambos cargados en memoria.

No hay que confundir procesos con archivos o programas. Por ejemplo, un compilador C no es un proceso, pero un compilador C ejecutándose, será un proceso para el sistema operativo y, por tanto, le asignará recursos (CPU, memoria, etc.) y controlará su ejecución.

Se utiliza una estructura denominada **bloque de control de procesos** para **identificar unívocamente cada proceso y controlar** todos los aspectos de **su ejecución**.

¿Cómo se ejecuta un proceso?

Es importante señalar que, **para que un proceso se ejecute, su secuencia de instrucciones debe encontrarse en la memoria principal**. Además, en todos los sistemas operativos modernos, se va intercalando la ejecución de distintos procesos, de forma que se alternan el uso del procesador.

Para saber **en qué posición de memoria se encuentra la siguiente instrucción que debe ejecutarse**, el procesador dispone de un **registro** llamado **contador de programa**, que irá cambiando de valor según pase el tiempo. La **secuencia de valores** que vaya teniendo el **contador de programa podrá apuntar a instrucciones de diferentes procesos.**

El **procesador ejecutará el código** perteneciente a un **módulo** del **sistema operativo**, llamado **Distribuidor** (en inglés, *Dispatcher*), **cada vez que un proceso haya consumido su tiempo** (medido en ciclos de instrucción) **o haya solicitado algún servicio por el que deba esperar** (p. ej. una operación de E/S) para **intentar ceder el procesador a otro proceso**.
Podemos definir un **ciclo de instrucción** como el **tiempo que emplea el procesador en ejecutar una instrucción en lenguaje máquina** y, de un modo simplificado, podríamos dividirlo en **dos pasos**:

- El **ciclo de lectura** (en inglés, *fetch*), que consiste en cargar una **instrucción desde la memoria principal a los registros del procesador**
- El **ciclo de ejecución** (en inglés, *execute*), que consiste en **interpretar la instrucción (decodificarla) y ejecutarla**, enviando las señales adecuadas a los componentes que deben realizar la operación que indica la instrucción.

Por este motivo, también suele llamarse **ciclo de *fetch-and-execute* o *fetch-decode-execute*.**

Llamamos **multitarea o multiprogramación** a la **capacidad** que tienen los **sistemas operativos** actuales de **alternar el uso del procesador entre distintos procesos**. Dada la velocidad a la que funcionan los procesadores, el usuario tiene la sensación de que los procesos se ejecutan al mismo tiempo.

Por otro lado, cuando **en un sistema informático disponemos de varios procesadores** (o incluso un **único procesador con varios núcleo**s), **pueden ejecutarse varios procesos al mismo tiempo**. A esta técnica la llamamos **multiproceso o multiprocesamiento**. Cuando **todos los procesadores (o núcleos) actúan en igualdad de condiciones, hablamos de multiproceso simétrico o SMP** (del inglés Symmetric Multi-Processing). Cuando el sistema dispone de **procesadores con funciones especializadas, hablamos de multiproceso asimétrico o AMP** (del inglés, Asymmetric Multi-Processing).

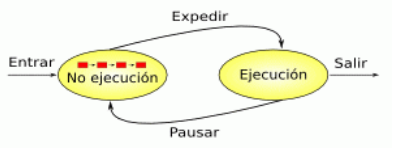
La **traza de un proceso** es la **secuencia ordenada de instrucciones que se ejecutan para dicho proceso**. Estudiando el modo en el que se intercalan las trazas de los diferentes procesos se puede estudiar el comportamiento general del procesador.

Estados de un proceso

El **principal trabajo del procesador** es **ejecutar** las **instrucciones de máquina** que se encuentran **en memoria principal**. **Para que un programa pueda ser ejecutado**, el **sistema operativo crea un nuevo proceso**, y el **procesador ejecuta una tras otra las instrucciones del mismo**.

En un entorno de multiprogramación, el procesador intercalará la ejecución de instrucciones de varios programas que se encuentran en memoria. El sistema operativo es el responsable de determinar las pautas de intercalado y asignación de recursos a cada proceso.

MODELO DE DOS ESTADOS

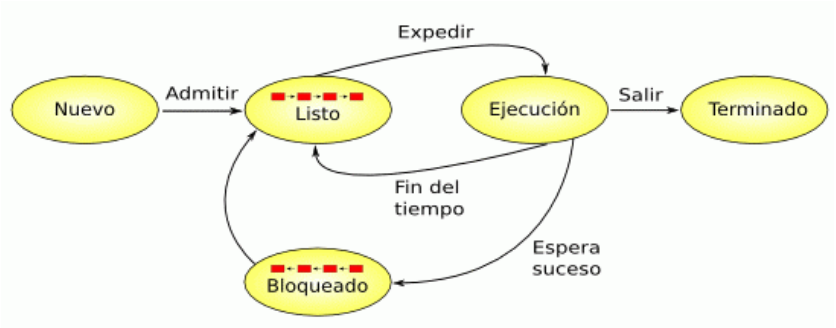


El **modelo de estados más simple es el de dos estados.** En este modelo, **un proceso puede estar ejecutándose o no**. Cuando se crea un **nuevo proceso**, se pone en **estado de No ejecución**. En **algún momento**, el **proceso que se está ejecutando pasará** al estado **No ejecución y se elegirá otro proceso de la lista** de procesos listos para ejecutar para ponerlo en **estado Ejecución**.

De esta explicación se desprende que es necesario que el sistema operativo pueda seguirles la pista a los procesos, conociendo su estado y el lugar que ocupa en memoria. Además, los **procesos que no se están ejecutando deben guardarse en algún tipo de cola mientras esperan su turno** para ejecutar.

MODELO DE CINCO ESTADOS

El **modelo anterior** de dos estados **funcionaría bien con una cola FIFO y planificación por turno** rotatorio para los procesos que no están en ejecución, si los procesos estuvieran siempre listos para ejecutar. **En la realidad, los procesos** utilizan datos para operar con ellos, y **puede suceder que no se encuentren listos, o que deban esperar algún evento antes de continuar**, como una operación de Entrada/Salida. Es por esto que se necesita un estado donde los procesos permanezcan bloqueados esperando hasta que puedan proseguir. **Se divide entonces al estado No ejecución en dos estados: Listo y Bloqueado. Se agregan además un estado Nuevo y otro Terminado.**



Los cinco estados de este diagrama son los siguientes:

1. **Ejecución: el proceso está actualmente en ejecución (posee todos los recursos, incluida la CPU).**
2. **Listo (o preparado): el proceso está listo para ser ejecutado, sólo está esperando** que el planificador así lo disponga y le entregue **la CPU**.
3. **Bloqueado (o suspendido): el proceso no se puede ejecutar hasta que no se produzca cierto suceso, como una operación de Entrada/Salida,** una señal de sincronización…
4. **Nuevo**: el **proceso se acaba de crear** y todavía no fue admitido por el sistema operativo. En general, los procesos que se encuentran en este estado todavía no fueron cargados en la memoria principal.
5. **Terminado**: el **proceso fue expulsado del grupo de procesos ejecutables**, ya sea porque terminó o por algún fallo, como un error de protección, aritmético, etc.

Algunos autores **consideran otro estado Inactivo**, sin embargo, este estado **representa procesos no conocidos por el sistema operativo, por lo que, estrictamente no es un estado del proceso dentro del S.O.**

Los nuevos estados Nuevo y Terminado son útiles para la gestión de procesos. En este modelo los **estados Bloqueado y Listo tienen ambos una cola de espera**. Cuando un nuevo proceso es admitido por el sistema operativo, se sitúa en la cola de 'listos'. A falta de un esquema de prioridades ésta puede ser una cola FIFO. Los procesos suspendidos son mantenidos en una cola de bloqueados. Cuando se produce el evento que estaban esperando, esos procesos se pasan a la cola de 'listos'.

Si existe un **esquema con diferentes niveles de prioridad de procesos** es conveniente mantener **varias colas de procesos listos, una para cada nivel de prioridad**, lo que ayuda a determinar cuál es el proceso que más conviene ejecutar a continuación.

PROCESOS EN ESPERA

Dos o más procesos pueden cooperar mediante señales de forma que **uno obliga a detenerse** a **los otros hasta que reciban una señal** para **continuar**. Se usa una **variable** llamada **semáforo para intercambiar señales**.

Si un **proceso está esperando una señal, se suspende (WAIT) hasta que la señal se envíe (SIGNAL).** Se mantiene una cola de procesos en ESPERA en el semáforo. La forma de elegir los procesos de la cola en ESPERA es mediante una política FIFO.

La **sincronización explícita entre procesos es un caso particular del estado "bloqueado".** En este caso, el **suceso que permite desbloquear un proceso no es una operación de entrada/salida, sino una señal generada** a propósito por el programador **desde otro proceso.**

Bloque de Control de Procesos

Para llevar a cabo la **gestión de un proceso**, es necesario que el **sistema operativo guarde** cierta **información** necesaria. El **sistema operativo almacena toda la información que necesita relativa a los procesos en el denominado Bloque de Control de Procesos**.

Cada vez que el **sistema operativo crea un nuevo proceso se genera el PCB** correspondiente que sirve de descripción en tiempo de ejecución durante el tiempo que dura el proceso. Los procesos son conocidos para el sistema operativo y, por tanto, elegibles para competir por los recursos del sistema sólo cuando existe un BCP activo asociado a ellos. Cuando el **programa termina, el BCP es eliminado para dejar espacio libre en el registro**, y usarlo para almacenar otros BCP.

El **bloque de control de procesos** difiere mucho de un sistema a otros, pero existen **contenidos** comunes:

- **Identificador del proceso**: **identifica de forma unívoca al proceso en el sistema**, generalmente se emplea un entero sin signo que se denomina PID (*Process IDentifier*)
- **Estado del proceso** para el planificador de procesos: preparado, activo, bloqueado…
- **Contador del programa**: dirección de la **siguiente instrucción a ejecuta**r.
- **Registros de la CPU**: contenidos **al final de la última ejecución** (contador de programa, puntero a pila, registros de datos, etc.).
- **Contexto de la ejecución**: valor de los registros del procesador, bits de estados, etc. Esto es, cada vez que se ejecuta el planificador y se realiza una **conmutación de procesos**, la **información sobre en qué lugar se encontraba la ejecución del proceso** se encuentra guardada aquí, **así como** el **lugar en el que se paró la** ejecución del **anterior proceso** (cada una en su respectivo BCP).
- **Información de planificación de la CPU**: prioridad, apuntadores a las colas, algoritmo usado…
- **Información contable y de identificación**: número de proceso, tiempo real y de CPU utilizado.
- **Información estado E/S**: solicitudes E/S pendientes, lista archivos abiertos, etc.
- **Aspectos relacionados con la administración de memoria**: tales como el espacio de direcciones y la cantidad de memoria asignada a un proceso.
- **Aspectos relacionados con la administración de ficheros**: tales como los ficheros con los que el proceso está actualmente operando.

- **Los procesadores en los que el proceso puede ejecutarse**: en caso de soportar el sistema multiprocesador.
- En el caso de un sistema operativo tipo UNIX: el **proceso padre** de dicho proceso **y** la relación de **procesos hijos.**
- **Estadísticas temporales**: tiempo de lanzamiento del proceso, tiempo en estado activo, etc.
- **Prioridad del proceso**: cada proceso tiene asignada una prioridad de forma que, en cualquier instante, el proceso que mayor prioridad tiene asignada, de entre los que están en estado Esperando, es el que se ejecutará.
- **Recursos asociados al proceso**: como ficheros, semáforos, etc.

El **sistema operativo crea listas de BCP agrupados por el estado de los procesos**: una lista para procesos preparados, otra de procesos bloqueados o suspendidos…

¿Cuándo acaba un proceso?

Todos los **sistemas operativos deben tener** un **mecanismo para identificar cuándo termina un proceso**. Si se trata de un script o un proceso por lotes concluirá cuando acaben sus instrucciones o cuando se encuentre una orden de parada. Si es un proceso interactivo, será el usuario el que elija el momento de terminar.

Además, un **proceso puede verse interrumpido abruptamente por diversos motivos**. Entre ellos, podemos encontrar los siguientes:

- **Sobrepasar el tiempo de ejecución** asignado al proceso (tiempo real, de uso del procesador, etc.) o el tiempo máximo de espera ante un suceso.
- No disponer de memoria suficiente para satisfacer las solicitudes del proceso
- **Que el proceso trate de acceder a posiciones de memoria o recursos** del sistema que **no** tiene **autorizados**.
- Que una de sus instrucciones contenga un **error aritmético o los datos no sean del tipo o tamaño adecuado**.
- Que surja un **error en una operación de entrada/salida** (no existe un archivo, se produce un error de lectura, etc.)
- Que **una instrucción del programa no exista en el juego de instrucciones** o que sea una instrucción reservada al sistema operativo.
- Que el **sistema operativo, el usuario o el proceso padre decida terminarlo**. También suelen terminar los procesos hijos cuando termina el proceso padre.

Lógicamente, **cuando un proceso termina, abandona su estado** (En ejecución, Preparado, Bloqueado) **y es eliminado de la cola** o colas que dependan del Distribuidor.

Planificación de procesos

Los sistemas operativos cuentan con un **componente** llamado **planificador**, **que se encarga de decidir cuál de los procesos hará uso del procesador**. La **toma de esta decisión**, así como el tiempo de ejecución del proceso, estará **dada por un algoritmo, denominado Algoritmo de Planificación**.

La **Planificación de procesos tiene como principales objetivos**:

- **Equidad**: **todos** los **procesos** deben ser **atendidos**.
- **Eficacia**: el **procesador** debe estar **ocupado el 100%** del tiempo.

- **Tiempo de respuesta**: el **tiempo empleado en dar respuesta a las solicitudes** del usuario debe ser **el menor posible**.
- **Tiempo de espera**: **reducir al mínimo el tiempo de espera de los resultados** esperados por los usuarios por lotes.
- **Rendimiento**: **maximizar el número de tareas** que se procesan **por** cada **hora**.

Tipos de planificación

Podemos hablar de **tres tipos principales de planificación**:

1. **A largo plazo**: sus objetivos son **añadir nuevos procesos al sistema, tomándolos de la lista de espera y dar al planificador a corto plazo una mezcla equilibrada de trabajos.** Este tipo de planificación era el más frecuente en los sistemas de lotes y multiprogramados en lotes; las **decisiones se toman considerando los requisitos pre-declarados** de los procesos y **los que el sistema tenía libres al terminar algún otro proceso**. La planificación a largo plazo puede llevarse a cabo con periodicidad de **una vez cada varios segundos, minutos e inclusive horas.**
   En los **sistemas interactivos**, casi la totalidad de los que se usan hoy en día, **este tipo de planificación no se efectúa**, dado que es normalmente el usuario quien indica expresamente qué procesos iniciar.
2. **A medio plazo**: decide **qué procesos es conveniente bloquear en determinado momento, sea por escasez/saturación de algún recurso** (como la memoria primaria) **o** porque están realizando alguna **solicitud que no puede satisfacerse momentáneamente**; se encarga de **tomar decisiones respecto a los procesos conforme entran y salen del estado de bloqueado** (aquellos que están a la espera de algún evento externo o de la finalización de transferencia de datos con algún dispositivo). **Añade o elimina procesos de memoria principal** modificando, por tanto, el grado de multiprogramación.
3. **A corto plazo (planificador de la CPU)**: decide **cómo compartir en todo momento la CPU entre todos los procesos que requieren de sus recursos**. La planificación a corto plazo se lleva a cabo **decenas de veces por segundo** (razón por la cual debe ser código muy simple, eficiente y rápido); es el **encargado de planificar los procesos que están listos para ejecución.**
   El **planificador a corto plazo puede ser invocado cuando un proceso** se encuentra en **algunas** de las **cuatro siguientes circunstancias**:
   1º. **Pasa de estar ejecutando a estar en espera** (por ejemplo, por solicitar una operación de E/S, esperar a la sincronización con otro proceso, etc.)
   2º. **Pasa de estar ejecutando a estar listo** (por ejemplo, al ocurrir una interrupción)
   3º. **Deja de estar en espera a estar listo** (por ejemplo, al finalizar la operación de E/S que solicitó)
   4º. **Finaliza su ejecución, y pasa de ejecutando a terminado**

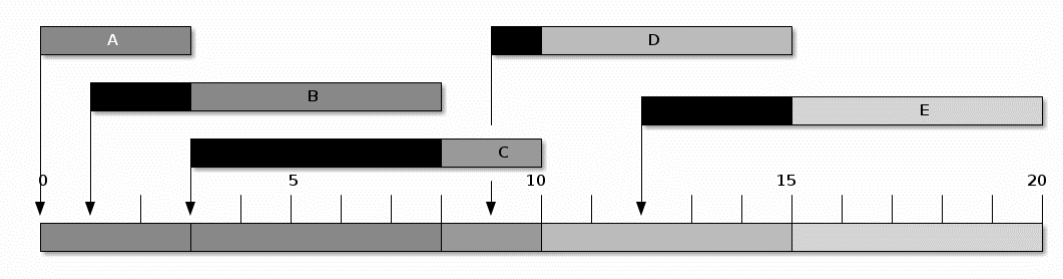
Algoritmos de planificación

Primero en llegar, primero en ser servido (*FCFS*)

El esquema más simple de planificación es el **Primero en llegar, primero en ser servido** (*First come, first serve, FCFS*). En este algoritmo **cada proceso se ejecuta en el orden en que llega, y mantiene el procesador hasta que finaliza su ejecución.** El 'despachador' es muy simple, básicamente una cola FIFO.

Consideremos los siguientes procesos:

Proceso	Tiempo de Llegada	*t*	Inicio	Fin	*T*	*E*	*P*
A	0	3	0	3	3	0	1
B	1	5	3	8	7	2	1.4
C	3	2	8	10	7	5	3.5
D	9	5	10	15	6	1	1.2
E	12	5	15	20	8	3	1.6
Promedio		4			6.2	2.2	1.74



Si bien un esquema FCFS **reduce al mínimo la sobrecarga administrativa** (que incluye tanto el tiempo requerido por el planificador para seleccionar al siguiente proceso, como el tiempo requerido para el cambio de contexto), el **rendimiento** percibido por los **últimos procesos** en llegar (o por procesos cortos llegados en un momento inconveniente) **resulta inaceptable.**

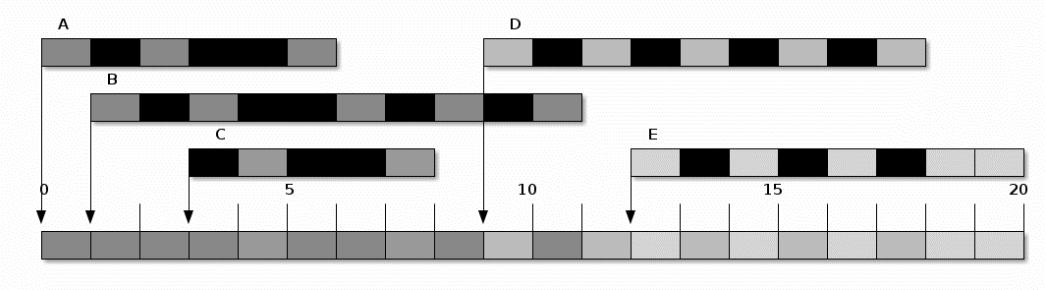
FCFS tiene características claramente inadecuadas para trabajo interactivo, sin embargo, al no requerir de hardware de apoyo (como un temporizador) sigue siendo ampliamente empleado

Ronda (*Round Robin*)

El esquema 'ronda' busca dar una **relación de respuesta buena, tanto para procesos largos, como para los cortos**. La principal diferencia entre la 'ronda' y FCFS es que en este caso **a cada proceso que esté en la lista de procesos listos, lo atenderemos durante un período de tiempo fijo o *quantum* (*q*).** Si **un proceso no ha terminado de ejecutar al final de su *quantum*, será interrumpido y puesto al final de la lista** de procesos listos, **para que espere a su turno** nuevamente. Los procesos que nos entreguen los planificadores a mediano o largo plazo se agregarán también al final de esta lista.

Con la misma tabla de procesos que encontramos en el caso anterior (y, por ahora, ignorando la sobrecarga administrativa provocada por los cambios de contexto), obtendríamos los siguientes resultados:

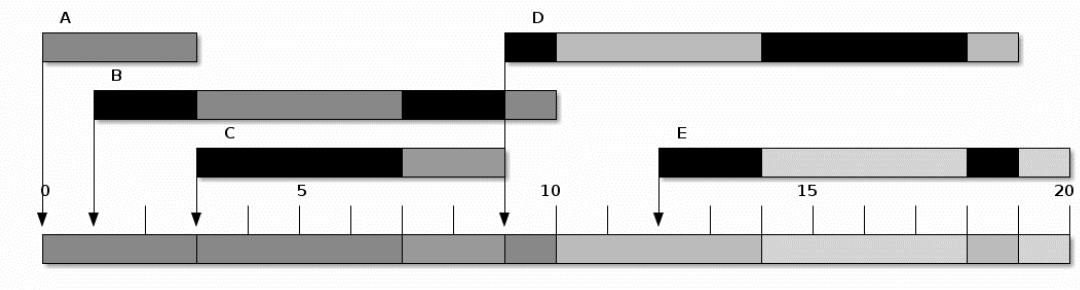
Proceso	Tiempo de Llegada	t	Inicio	Fin	T	E	P
A	0	3	0	6	6	3	2.0
B	1	5	1	11	10	5	2.0
C	3	2	4	8	5	3	2.5
D	9	5	9	18	9	4	1.8
E	12	5	12	20	8	3	1.6
Promedio		4			7.6	3.6	1.98



Ronda (*Round Robin*), con q=1

La *ronda* puede ser ajustada modificando la duración de *quantum*. **Conforme incrementamos** el *quantum*, **la ronda tiende a convertirse en FCFS.** Si cada *quantum* es arbitrariamente grande, todo proceso terminará su ejecución dentro de su *quantum*; por otro lado, conforme decrece q, mayor frecuencia de cambios de contexto tendremos. Si repetimos el análisis anterior bajo este mismo mecanismo, pero con un *quantum* de 4 *ticks*, tendremos:

Proceso	Tiempo de Llegada	t	Inicio	Fin	T	E	P
A	0	3	0	3	3	0	1.0
B	1	5	3	10	9	4	1.8
C	3	2	7	9	6	4	3.0
D	9	5	10	19	10	5	2.0
E	12	5	14	20	8	3	1.6
Promedio		4			7.2	3.2	1.88



Ronda (*Round Robin*), con q=4

Si bien aumentar el *quantum* mejora los tiempos promedio de respuesta, aumentarlo hasta convertirlo en un FCFS efectivo degenera en una penalización a los procesos cortos. Silberschatz apunta a que, en general, **el quantum debe mantenerse inferior a la duración promedio del 80% de los procesos.**

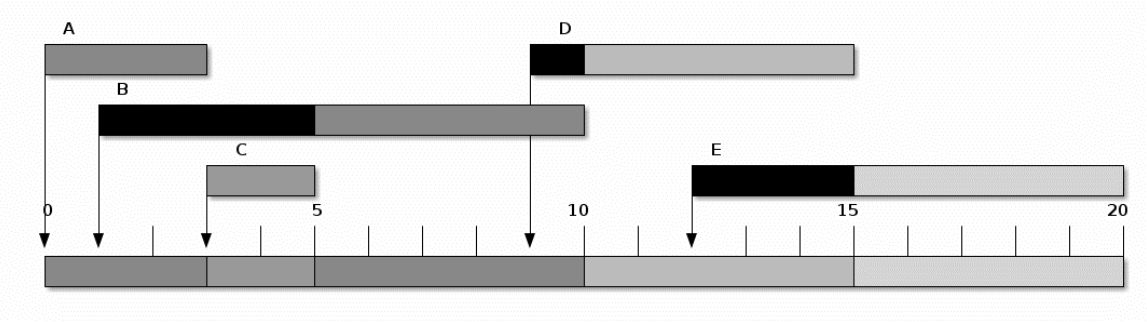
El proceso más corto a continuación (SPN)

Cuando no tenemos la posibilidad de implementar multitarea preventiva, pero requerimos de un algoritmo más justo, y **contamos con información por anticipado del tiempo que requieren los procesos que forman la lista, podemos elegir el más corto** de los presentes.

Ahora bien, es muy difícil contar con esta información antes de ejecutar el proceso. Es más frecuente buscar caracterizar las necesidades del proceso: ver si durante su historia de ejecución ha sido un proceso tendiente a manejar ráfagas limitadas por entrada-salida o limitadas por procesador, y cuál es su tendencia actual.

Empleando el mismo juego de datos de procesos que hemos venido manejando como resultados de las estimaciones, obtendríamos el siguiente resultado:

Proceso	Tiempo de Llegada	*t*	Inicio	Fin	*T*	*E*	*P*
A	0	3	0	3	3	0	1.0
B	1	5	5	10	9	4	1.8
C	3	2	3	5	2	0	1.0
D	9	5	10	15	6	1	1.2
E	12	5	15	20	8	3	1.6
Promedio		4			5.6	1.6	1.32



El proceso más corto a continuación (SPN)

Como era de esperarse, **SPN favorece a los procesos cortos**. Un proceso largo puede esperar mucho tiempo antes de ser atendido. En un sistema poco ocupado, en que la cola de procesos listos es corta, SPN generará resultados muy similares a los de FCFS.

Menor tiempo restante a continuación (SRTN)

El **siguiente proceso a ejecutar será al que le quede menor tiempo de ejecución**. Este tipo de planificación es óptima, **cuando se conocen** con antelación **los futuros tiempos de ejecución** de los procesos.

Planificación por prioridad fija

En este tipo de planificación **a cada proceso se le asigna una prioridad siguiendo un criterio determinado, y de acuerdo con esa prioridad será el orden en que se atienda cada proceso**.

Planificación garantizada

Para realizar esta **planificación el sistema tiene en cuenta el número de usuarios que deben ser atendidos.** Para un número **"n"** de **usuarios** se asignará **a cada uno un tiempo de ejecución igual a 1/n.**

Planificación de Colas Múltiples
El nombre se deriva de **MQS** (***Multilevel Queue Schedulling***). En este algoritmo la **cola de procesos** que se encuentran en estado de **listos es dividida en un número determinado de colas más pequeñas**. Los **procesos** son **clasificados** mediante un **criterio** para determinar en qué cola será colocado cada uno cuando quede en estado de listo. **Cada cola puede manejar un algoritmo de planificación diferente a las demás.**

Sincronización de procesos

Dentro de los Sistemas Operativos los procesos que se ejecutan son muy variados, algunos utilizan la CPU para realizar de lectura de memoria, u otros campos que competen a la aplicación. Otros pueden intentar modificar los datos que leen. Pero, ¿**qué sucede si dos o más procesos tienen que hacer estas operaciones simultáneamente? Es decir, por ejemplo, modificar los datos que un proceso está leyendo.**

Pues es aquí donde puede 'provocarse un caos', pues la información podría ser modificada sin previo aviso o suceder acciones que no estaban planteadas. Para esto se plantea la **Sincronización de Procesos**.

Esto **consiste en utilizar estructuras algorítmicas que permitan implementar el control de los procesos o hilos que se van a ejecutar de acuerdo a condiciones dadas, para que no se produzca situaciones como Bloqueos, Bloqueos mutuos o accesos a secciones críticas** del programa **que no pueden ser accedidas en ese momento.**

Un **sistema operativo multiprogramado** es un **caso particular de sistema concurrente** donde **los procesos compiten por el acceso a los recursos compartidos o cooperan dentro de una misma aplicación para comunicar información**. Ambas situaciones son tratadas por el sistema operativo mediante **mecanismos de sincronización** que **permiten el acceso exclusivo** de forma coordinada **a los recursos y a los elementos de comunicación compartidos.**

Según el modelo de sistema operativo descrito anteriormente, basado en colas de procesos y transiciones de estados, los procesos abandonan la CPU para pasar a estado bloqueado cuando requieren el acceso a algún dispositivo, generalmente en una operación de E/S, pasando a estado preparado cuando la operación ha concluido y eventualmente volver a ejecución. La gestión de estos cambios de estado, es decir, **los cambios de contexto, es un ejemplo de sección crítica de código dentro del sistema operativo que debe ser ejecutada por éste en exclusión mutua**. Otros ejemplos de código que debe protegerse como sección crítica incluyen la programación de los dispositivos de E/S y el acceso a estructuras de datos y buffers compartidos.

Es necesario utilizar mecanismos de sincronización explícitos para garantizar acceso exclusivo a las variables compartidas y evitar situaciones de bloqueo. Puede producirse una condición de bloqueo sobre una variable cuando varios procesos acceden concurrentemente a la variable para actualizarla

SECCIÓN CRÍTICA

El modelo de sección crítica que vamos a utilizar sigue el siguiente protocolo genérico:

```
Entrar_SC(esta_SC) /* Solicitud de ejecutar esta_SC */
```

```
        /* código de esta_SC */
    Dejar_SC(esta_SC) /* Otro proceso puede ejecutar esta_SC */
```

Es decir, cuando un proceso quiere entrar a la sección crítica:

1. Ejecuta Entrar_SC(), y si la sección crítica está ocupada el proceso espera;
2. Ejecuta la sección crítica;
3. Ejecuta Dejar_SC(), permitiendo que entre uno de los procesos en espera.

Propiedades del acceso exclusivo a secciones críticas

Como **criterios de validez de un mecanismo de sincronización** nos referiremos al **cumplimiento de las siguientes condiciones**:

1. **Exclusión mutua: no puede haber más de un proceso simultáneamente** en la SC.
2. **No debe existir interbloqueo: ningún proceso fuera de la SC puede impedir que otro entre** a la SC.
3. **No se permite la espera indefinida: un proceso no puede esperar por tiempo indefinido** para entrar a la SC.
4. **Independencia del hardware: no se pueden hacer suposiciones acerca del número de procesadores o de la velocidad relativa de los procesos**.

SEMÁFOROS

Una abstracción más general es el semáforo, que **permite, sobre la base de las primitivas de dormir y despertar, almacenar los eventos ya producidos y despertar un único proceso bloqueado cuando se produce un evento pendiente**.

Un semáforo lleva asociada una **cola de procesos bloqueados** en él **y una cuenta de señales de despertar recibidas s**, lo que permite su utilización general para gestionar recursos, como vamos a ver.

Se definen dos operaciones atómicas sobre un semáforo, s:

```
espera(s) – también: p(s), down(s), wait(s), ...
señal(s) – también: v(s), up(s), signal(s), ...
```


Cuando un proceso ejecuta espera(s), si la cuenta asociada a s es mayor que cero el proceso continúa y la cuenta se decrementa; en caso contrario, el proceso se bloquea. Cuando un proceso ejecuta señal(s), se incrementa la cuenta; si hay procesos bloqueados despierta a uno.

Un **semáforo cuya variable solo puede tomar los valores 0 y 1 se llama semáforo binario**.

MONITORES

Son **módulos que encierran los recursos o variables compartidas como componentes internos privados y ofrece una interfaz de acceso a ellos** que **garantiza** el régimen de **exclusión mutua**.

La **declaración de un monitor incluye**:

- **Declaración de las constantes, variables, procedimientos y funciones** que son **privados** del monitor (solo el monitor tiene visibilidad sobre ellos).