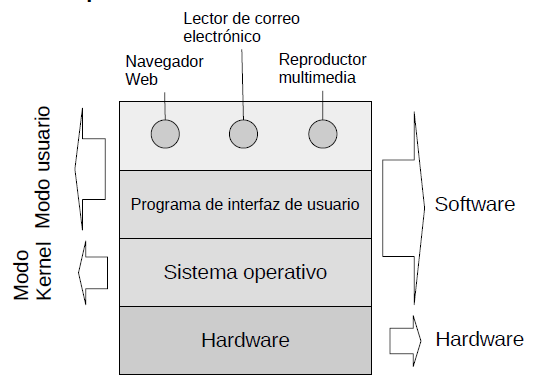
**Resumen Parcial 1 SO**



**Modo kernel/supervisor y modo usuario:**

El sistema operativo se ejecuta en **modo kernel**, dado que debe acceder y controlar todo el hardware de una computadora.

El resto de los programas de computadoras se ejecuta en **modo usuario**.

**Definición del concepto de sistema operativo:**

Según Stallings, un sistema operativo es un programa que controla la ejecución de aplicaciones y programas, y que actúa como interfaz entre las aplicaciones y el hardware del computador.

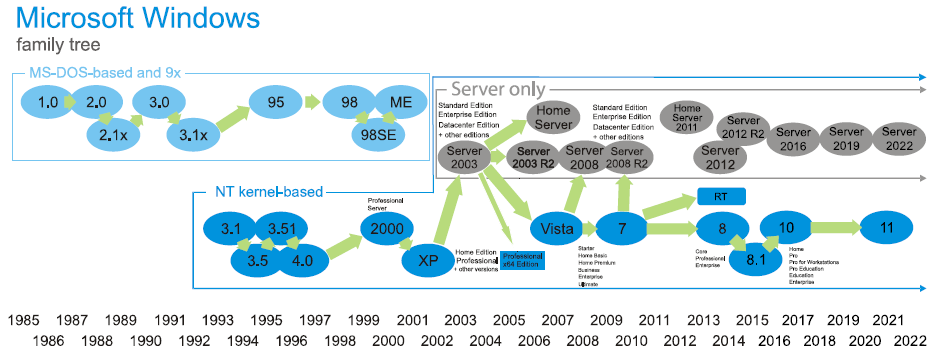
**SO como maquina extendida y como administrador de recursos:**

* Como maquina extendida: La arquitectura (conjunto de instrucciones, organización de memoria, E/S y estructura de bus) de la mayoría de las computadoras a nivel de lenguaje maquina es primitiva y compleja de programar.
* Como administrador de recursos: Una de las funciones del sistema operativo es crear buenas abstracciones para después implementar y administrar los objetos abstractos entonces creados.

Otra es proporcionar una asignación ordenada y controlada de los procesadores, memorias y dispositivos de E/S, entre los diversos programas que compiten por estos recursos. Incluye el multiplexaje (compartir) de recursos en dos formas distintas: en el tiempo (micro con varios procesos por ejecutar) y en el espacio (memoria que contiene varios programas).

**Historia de los SO y las computadoras:**

1. 1945 -> Tubos al vacío: Lenguajes de programación y SO desconocidos.
2. 1955 -> Transistores y sistemas por lotes: Aparecen los mainframes y lenguajes de prog.
3. 1965 -> Circuitos integrados y multiprog: Aparece el SO OS/360, multiprogramación (partición de memoria). Spooling de impresión. Minicomputadoras.
4. 1980 -> Computadoras personales (microcomputadoras): Bill Gates compra DOS (Disk Operating Sistema). 1983 sale la IBM PC/AT, con CPU Intel 80286 y MS-DOS.
5. 1990 -> Computadoras móviles: Al inicio predominaba Symbian, Blackberry OS de RIM. Windows Phone llego tarde. Actualmente Android de Google es el SO dominante. iOS de Apple como segunda alternativa dominante.



**Historia de GNU/Linux y el software libre:**

* IMB se posicionaba como el principal fabricante de super ordenadores. **Se compraba un equipo (hardware) y venia incluido el software** (Se adquirían de forma conjunta) y también se recibía el código fuente del software. Se podría decir que el software era libre en la década del 60.
* 1969 IBM dio aviso de que sus programas vendrías por separado (comienzo valor propio al SW). Desde entonces se hizo normal que los fabricantes restringieran el acceso a los programas de pc
* 10 años después comenzó a surgir el movimiento de software libre de la mano de Richard Stallman.
* Unix fue desarrollado por Thompson y Ritchie en los Bell Labs de AT&T. Entre 1973 y 1974, Unix llego a varias universidades y centros de investigación. Pero UNIX tenía una licencia con restricciones de uso y solo con fines educativos. Con el código fuente se podía estudiar, mejorar y ampliar.

Al mismo tiempo, aparece una comunidad de desarrolladores de la Universidad de California, en Berkeley. Esta comunidad desarrollo su propia cultura y que fue muy relevante en la historia del software libre.

Unix sirvió como un ensayo para lo que luego fue GNU y Linux.

* A finales de los 70 y principio de los 80, AT&T realizo cambios en el acceso a las nuevas versiones. Paso a ser muy oneroso y difícil de conseguir.
* En 1991, AT&T demanda a la Universidad de California de Berkeley por publicar código de Unix BSD que el CSRG (Grupo de Investigación en Ciencias de la Computación) había desarrollado.
* En 1984, Richard Stallman renuncia al Al Lab del MIT para comenzar con el proyecto GNU.
* La idea de RMS era desarrollar un sistema de software completo de propósito general, pero completamente libre. GNU es un acrónimo recursivo “GNU’s not Unix”.
* Despues de 6 años de la creación de GNU ya faltaba muy poco para cerrar el sistema de software completo. Solo faltaba el núcleo (kernel) del sistema.
* Para Unix BSD apareció el 386BSD. Y para el proyecto GNU llego Linux.
* En 1991, Linux Torvalds propone hacer un sistema similar a Minix y libera una primera versión (0.01) del núcleo Linux.
* 1992 surgen las primeras distribuciones de GNU/Lunix.

**Arranque del hardware de computadora. Arranque de la computadora:**

1. Se enciende la Computadora.
2. Se ejecuta el Power On Selt Test (POST) del Basic Input Output System (BIOS).   
   Prueba la memoria RAM, Prueba el teclado y otros dispositivos básicos, Explota los buses   
   para detectar dispositivos conectados.
3. Se busca un dispositivo de arranque (Secuencia de arranque en el BIOS).  
   Se lee la lista de dispositivos de arranque de la CMOS.
4. Se lee el primer sector del disco en memoria (los primeros 512 bytes – MBR).Se examina la tabla de particiones, Se determina que partición esta activa, Se lee un cargador secundario de arranque que deberá leer el SO.
5. Se selecciona un Sistema Operativo y se procede a la carga del mismo.

**Tipos de Sistemas Operativos:**

* **Mainframe (IBM | UNIX | GNU/Linux):  
   -**Equipos que ocupan un cuarto completo.  
   -Mayor capacidad de E/S, poseen 1000 discos conectados.  
   -Se utilizan como servidores Web de alto rendimiento y comercio electrónico.  
   -Ejemplos de sistemas operativos: OS/360, OS/390, UNIX y GNU/Linux.
* **Servidores (Solaris | FreeBSD | GNU/Linux | Windows Server):** -Son computadoras personales grandes, estaciones de trabajo o mainframe.  
   -Brindan servicios en la red, compartiendo recursos c/ archivos e impresoras o Web.  
   -Muy utilizadas por los proveedores de Internet.
* **Multiprocesadores (GNU/Linux | Windows Server):** -Se ejecutan en computadoras en paralelo, multicomputadoras o multiproocesadores  
   -Generalmente, son variaciones de los sistemas operativos de servidores.  
   -Hoy existen computadoras personales con chips multinúcleos.  
   -Dada la cantidad de trabajos científicos en los sistemas de multiprocesadores se   
   hace fácil la implementación de chips multinúcleos.
* **Computadoras Personales (GNU/Linux | Windows | Macintosh):** -Soportan la multiprogramación.  
   -Proporcionan buen soporte para los usuarios.  
   -Generalmente, se utilizan como procesadores de texto, planilla e Internet.
* **Computadoras de bolsillo (Symbian OS | Blackberry OS | Android | iOS ):** -Samsung, Sony, Motorola y Nokia utilizaban Symbian OS.  
   -Luego aparece Blackberry OS de RIM.  
   -Actualmente, Android de Google es el sistema operativo dominante.  
   -iOS de Apple como segunda alternativa dominante.  
   -Proporcionan servicios de telefonía, fotográfica digital y otras funciones.
* **Integrados (GNU/Linux | QNX | VxWorks):** -Controlan dispositivos específicos (Tv’s, reproductores MP3 y grabadoras DVD).  
   -Se los conoce como sistemas embebidos o incrustados.  
   -No aceptan la instalación de nuevos programadas por parte del usuario final.
* **Nodos sensores (TinyOS):**  
   ­-Son pequeñas computadoras que se comunican entre si con una estación base.  
   -Los sensores son pequeñas computadoras con radios integrados.  
   -Cada nodo sensor es una verdadera computadora, con sensores ambientales.  
   -El SO debe ser pequeño y simple debido a que los nodos tienen escasos recursos.
* **Tiempo real (e-Cos | FreeRTOS):** -Se caracterizan por tener el tiempo como un parámetro clave.  
   -SO de TR duros que se encuentran en el control de procesos industriales y en la aeronáutica.  
   -Los sistems de audio digital, de multimed y los de teléfonos, son SO de TR suaves.  
   -Los sistemas para computadoras de bolsillos, sistemas integrados y sistemas en tiempo real se traslapan en fomra considerable.
* **Tarjetas inteligentes:**  
   -Son dispositivos del tamaño de una tarjta de crédito que contienen un chip de CPU.  
   -Poseen severas restricciones de poder de procesamiento y de memoria.  
   -Normalmente, se lo utiliza para realizar pagos electrónicos.  
   -El sistema operativo es en extremo primitivo.

**Conceptos de los sistemas operativos:**

* **Proceso:** Es un programa en ejecución. Se lo considera un recipiente que guarda toda la información necesaria para ejecutar programas.
* **Espacio de direcciones:** Es una abstracción creada por el SO, que se utiliza como el conjunto de direcciones al que puede hacer referencia un proceso.
* **Archivos:** Concepto clave de casi todos los sistemas operativos es el sistema de archivos.  
   (Ocultar las peculiaridades, Direct raíz, Direct de trabajo, Tipos de archivos).
* **Entrada/Salida:** Todas las computadoras tienen dispositivos físicos para adquirir entrada y producir salida.

Partir del software de E/S es independiente de los dispositivos. Otras partes del software, como los controladores de dispositivos, son específicas para ciertos dispositivos de E/S.

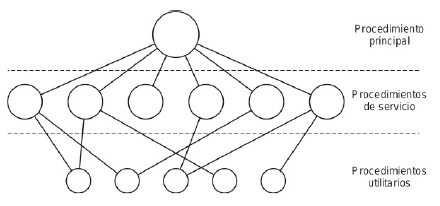
* **Protección:** Es responsabilidad del SO administrar la seguridad del sistema de manera que los archivos, por ejemplo, solo sean accesibles para los usuarios autorizados.
* **Shell:** Es el intérprete de comandos de UNIX o GNU/Linux. Es la interfaz principal entre un usuario sentado en su terminal y el SO, a menos que el usuario este usando una interfaz gráfica de usuario.

El SO es el código que lleva a cabo las llamadas al sistema.  
Los editores, computadores, ensambladores, enlazadores e intérpretes de comandos en definitiva no forman parte del SO, aun cuando son importantes y útiles.

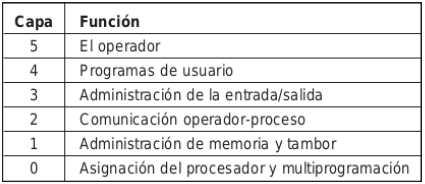
**Llamadas al sistema:**

Es la interfaz fundamental entre una aplicación y el núcleo de Linux.  
Existen llamadas al sistema para: Administrar procesos, archivos y sistema de directorios y archivos.

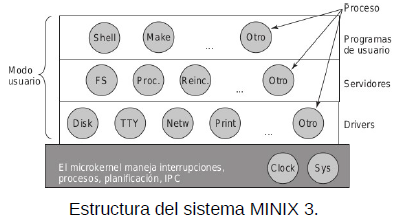
**Diseño de Sistemas Operativos:**

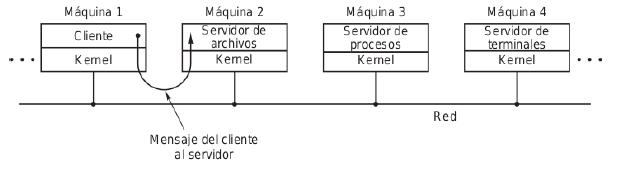


* **Monolíticos:** Es el diseño más común, todo   
  el sistema operativo se ejecuta como un solo programa modo kernel. (SO: Procedimiento principal, de servicio y utilitarios).



* **Capas:** Organiza el SO como una jerarquía de capas, cada una construida encima de la que   
  tiene debajo.



* **Microkernels:** El diseño de este es lograr una alta confiabilidad al dividir el SO en módulos pequeños y bien definidos, solo uno de los cuales (el microkernel) se ejecutan en modo kernel y el resto se ejecuta como procesos de usuario ordinarios, sin poder relativamente.
* **Cliente/Servidor:** Es diferencias dos clases de procesos: los servidores, cada uno de los cuales proporciona cierto servicio, y los clientes, que utilizan estos servicios.
* **Máquinas virtuales:** Es un entorno virtual que funciona como una computadora dentro de una computadora. Se ejecuta en una partición aislada de su computadora host con sus propios recursos de potencia de CPU, memoria, un sistema operativo (Windows, Linux, MacOS) y otros recursos.

(a) Hipervisor tipo 1 | (b) hipervisor tipo 2.

(a) denominados <bare metal>: se ejecutan directamente sobre el hardware del host.

(b) denominados <alojados>: se ejecutan como capa de software sobre un SO.

Fin Unidad 1

Unidad 2

PROCESOS

**Concepto de proceso:**

Un proceso es una abstracción de un programa en ejecución, que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

**Creación de procesos:**

Existen 4 eventos principales que producen la creación de procesos en un SO.

1. El arranque del sistema.
2. La ejecución, desde un proceso, de una llamada al sistema para la creación de procesos.
3. Una petición de usuario para crear procesos.
4. El inicio de un trabajo por lotes.

**Jerarquías de procesos:**

Algunos SO, organizan sus procesos de forma jerárquica, por lo tanto, existe un proceso padre que crea sus procesos hijos, y esos procesos hijos también pueden crear más procesos dentro de la jerarquía, este es el caso de los sistemas tipos UNIX.

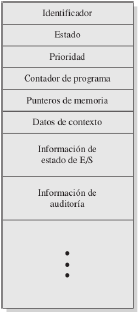
**Terminación de procesos:**

Tarde o temprano un nuevo proceso terminará su ejecución, y en general es debido a las siguientes condiciones:

1. Salida normal (voluntaria).
2. Salida por error (voluntaria).
3. Error fatal (involuntaria).
4. Eliminado por otro proceso (involuntaria).

**Elementos característicos de un proceso y su implementación:**

La implementación del modelo de procesos, invita a que el SO mantenga una tabla (arreglo de estructuras) llamada **Tabla de procesos o Bloque de control de programa (BCP)**, que tenga una sola entrada por cada proceso.



* **Identificador:** es un valor número único de proceso.
* **Estado:** indica el estado actual de un proceso.
* **Prioridad:** nivel de prioridad.
* **Contador de programa:** dirección de la siguiente instrucción.
* **Puntero a memoria:** al código del programa y los datos.
* **Datos de contexto:** datos en los registros del microprocesador.
* **Información de estado E/S:** peticiones de E/S y lista de archivos.
* **Información de auditoria:** cantidad de tiempo en microprocesador.

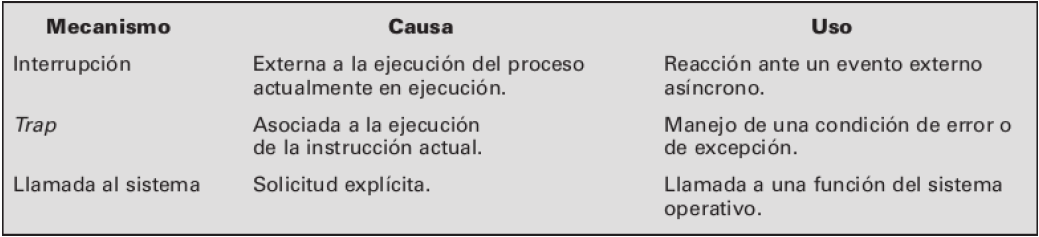
**Tipos de procesos:**

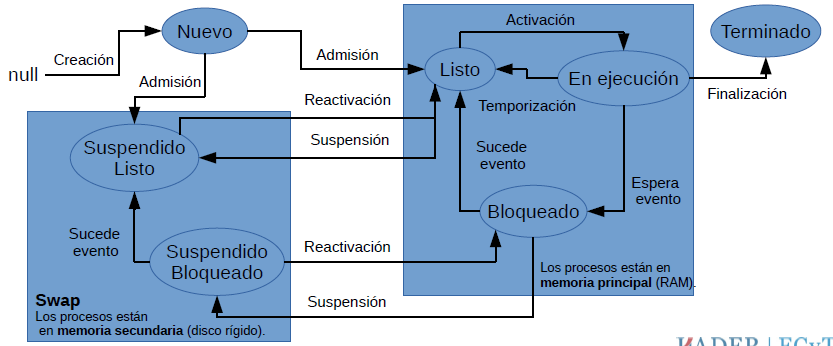
Podemos clasificarlos según:

* **Diseño:**○ **Reutilizables:** Pueden cambiar los datos que utilizan y se cargan en memoria cada vez que se usan. Ejemplo: Writer/Word, Calc/Excel, etc.  
    
  ○ **Reentrantes:** No tienen asociados datos, solo poseen código puro y se carga una sola copia del código en memoria. Ejemplo: Librerías compartidas, editores de código fuente.
* **Acceso a los recursos (CPU):**○ **Apropiativos:** Acceden a los recursos y solo los abandonan de forma voluntaria y no dejan que otros procesos lo utilicen.  
    
  ○ **No apropiativos:** Permiten a otros procesos apropiarse de los recursos que se encuentran actualmente en uso.
* **Permanencia en memoria:**○ **Residentes:** Permanecen en memoria todo el tiempo que dure su ejecución (desde su creación hasta su terminación).
* ○ **Intercambiables** **(swappable):** Es lo más habitual, dado que el SO puede decidir llevarlos a disco para liberar memoria que otro proceso necesita ocupar.

**Cambio de contexto:**

El cambio de contexto se da cuando dos o más procesos tienen que compartir tiempo en el microprocesador. Este algoritmo se lleva a cabo guardando el estado de un proceso en un BCP luego de que el mismo cumplió su tiempo de ejecución o por que un proceso con mayor prioridad necesita hacer uso del micro.

* El sistema almacena el estado del proceso en ejecución anterior en un BCP, carga el BCP del siguiente proceso a ejecutar y le asigna la CPU.



Prioridades

Se pueden clasificar en (1) **asignadas por el SO** y (2) **asignadas por el propietario**.

1. Asignadas en el momento de comenzar su ejecución. Dependen de los privilegios de su propietario y el modo de ejecución.
2. Es el propietario quien asigna a cada proceso la prioridad con que este debe ejecutarse (comando nice y renice).

Prioridades en el **kernel** de **Linux**, se relacionan con las clases de planificadores de proce/hilos:

* **Planificación FIFO o FCFS,** en tiempo real.
* **Planificación circular RR,** en tiempo real.
* **Planificación de tiempo compartido.**

**Niveles de prioridades para los procesos/hilos en el kernel de Linux:**

* Los procesos/hilos en **tiempo real** se representan en forma interna con los niveles de prioridad de **0 a 99** (0 max y 99 min prioridad).
* Los procesos/hilos convencionales que no son de tiempo real se asocian con los niveles de prioridad del **100 al 139**.

Es decir, Linux, los diferencia de manera interna entre **140 niveles** de prioridad (0 a 99 tiempo real, 100 a 139 no tiempo real).

La mayoría de los sistemas UNIX y Linux asocia un buen valor con cada proceso/hilo.

El predeterminado es **0**, pero se puede modificar si utilizamos la llamada al sistema **nice**(valor), donde el valor puede estar entre **-20 y 19**. Este valor determina la prioridad estática de cada proceso/hilo.

Ejemplo de uso, os, top, nice y renice.

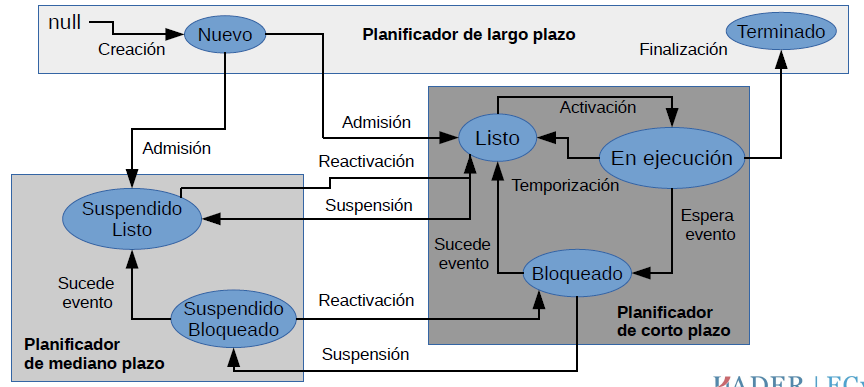
Planificador de procesos

Es el módulo del SO que realiza la función de seleccionar un proceso en estado de Listo que pasa a estado de ejecución.

Su **objetivo principal** es dar un buen servicio a todos los procesos que existan en un momento dado en el sistema.

**Objetivos secundarios**: justicia, máxima capacidad de ejecución, máximo número de usuarios inactivos, predecibilidad, minimización de la sobrecarga (overhead), equilibrio en el uso de recursos, seguridad de las prioridades.

* Niveles de planificación: corto, mediano y largo plazo

****

**Criterios a tener en cuenta:**

* Tiempo: De respuesta.  
   De servicio o retorno (t final – t inicial).  
   De ejecución.  
   De procesador (t de ejecución sin considerar E/S).  
   De espera (t de servicio – t de ejecución).

De núcleo (decisiones de planificación y cambios de contexto).

De inactividad (momento donde el micro no tiene un proceso en ejecución).

* Eficiencia (índice medio de servicio) (tiempo de ejecución / tiempo de servicio).
* Rendimiento. (cantidad de trabajos procesados sobre unidad de tiempo)

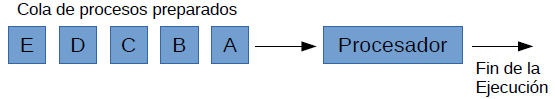
**Políticas de planificación:**

* **Apropiativa o con expulsión** (nonpreemptive). Son las que producen un cambio de proceso con cada cambio de contexto; es decir, el proceso que está haciendo uso del procesador puede ser temporalmente suspendido y permitir que otro proceso se apropie del procesador. Se utilizan en SO con tiempo compartido y tiempo real.
* **No apropiativas o sin expulsión** (preemptive). Son aquellas en las que un proceso no abandona nunca el procesador desde su comienzo hasta su fin. Se utilizan en sistemas de proceso por lotes

Algoritmos de planificación

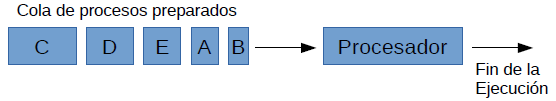
**First Come, First Served (FCFS)** (Primero en llegar, primero en ser servido):

* No es apropiativa.
* Es justa, aunque los procesos largos hacen esperar mucho a los cortos.
* Es una política predecible.
* El tiempo medio de servicio varía mucho en función del numero de procesos y su duración.

****

**Shortest Job Next (SJN)** (El proceso más corto a continuación):

* No es apropiativa.
* El tiempo de espera aumenta de acuerdo con la longitud de los procesos.
* Es poco predecible.
* No es justa con los procesos largos (sufren inanición /starvation).
* Buen tiempo de servicio.

****

**Highest Response Ratio Next (HRRN)** (Primero el de mayor tasa de respuesta)

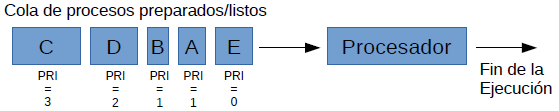
* No es apropiativa.
* Es justa (considera la edad del proceso).
* Es costosa de poner en práctica.
* Produce una sobrecarga en el sistema.
* Corrige algunas injusticias de SJN y FCFS.

R = ( w + s ) / s -> R = tasa de respuesta.  
 w = tiempo invertido esperando por el procesador.

s = tiempo de servicio esperado.

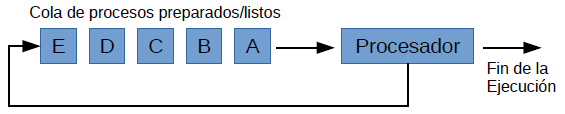
**Prioridad sin expulsión** (Primero el proceso con mayor prioridad)

* No es apropiativa, pero se puede implementar como apropiativa.
* No es justa con procesos de baja prioridad.
* Las prioridades pueden ser asignadas de forma interna por el SO o de forma externa.



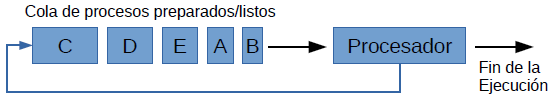
**Round-Robin (RR)** (Cola circular) (q = n°)

* Es apropiativa.
* Baja sobrecarga si el cambio de contexto es eficiente.
* Es la política más utilizada para tiempo compartido.
* Ofrece un índice de servicio uniforme para todos los procesos.



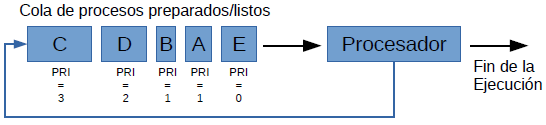
**Shortest Remaining Time (SRT)** (Próximo proceso, el tiempo restante más corto).

* Es muy eficiente
* Es una variante de SJN, para hacerla apropiativa.
* Puede ser injusta, un proceso largo que puede ser postergado por otro corto.
* Presenta una mayor recarga.
* Excelente tiempo medio de servicio.

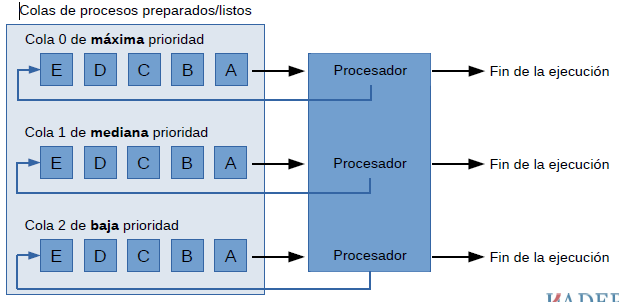


**Prioridad con expulsión** (Primero el proceso con mayor prioridad).

* Es apropiativa, pero se puede implementar como no apropiativa.
* Es injusta para los procesos de baja prioridad (se mitiga con envejecimiento).
* Las prioridades pueden ser asignadas de forma interna por el SO o externa.



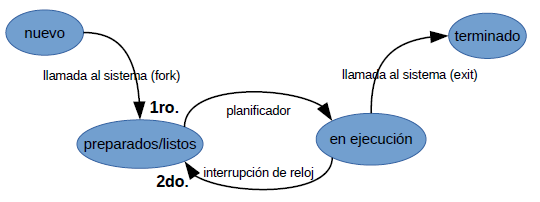
**Colas multinivel**

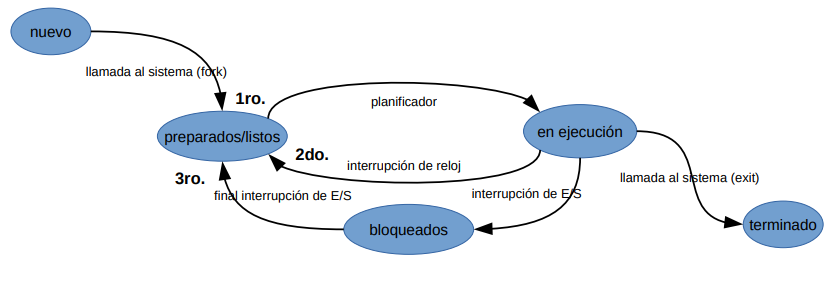
****

**RESUMEN DE LAS POLITICAS DE PLANIFICACION:**

* Políticas **apropiativas o con expulsión** (nonpreemptive).  
   Shortest Remaining Time (SRT).  
   Round Robin (RR)  
   Prioridad  
   Multinivel
* Políticas **no apropiativas o sin expulsión** (preemptive).  
   First Come, First Served (FCFS)  
   Shortest Job Next (SJN)  
   Highest Response Ratio Next (HRRN)  
   Prioridad  
   Multinivel

**Simultaneidad de eventos en el corto plazo para procesos preparados/listos:**





**Para llevar a cabo los ejercicios, se prioriza la llamada al sistema (fork)** antes que la interrupción de reloj. **En la vida real, el orden es el inverso**, siempre se prioriza la interrupción de reloj respecto de cualquier llamada al sistema u otra interrupción de entrada/salida.

Llega hasta el archivo 7 de la unidad 2

**Algoritmo de planificación de colas multinivel con retroalimentación**

Colas multinivel con retroalimentación

*Características:*

♦ Favorece a los procesos cortos (similar a SJN)

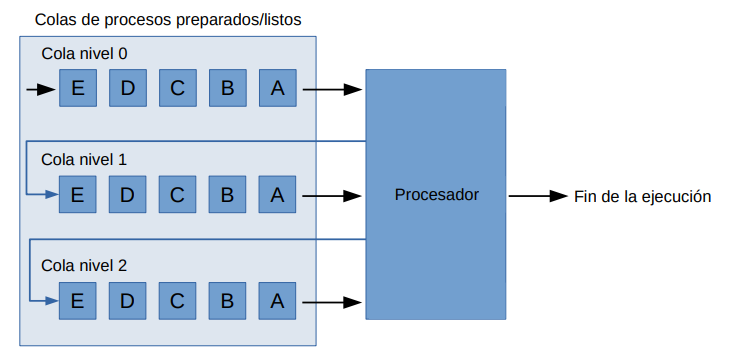
♦ Se implementa con RR o tiempo compartido

♦ Existe desalojo por rodaja de tiempo (baja su prioridad)

♦ Aumenta su rodaja de tiempo cuando se baja la prioridad del proceso

♦ Mayor tiempo de ejecución se le otorga menos prioridad al proceso

Ejemplo:



**Concepto de proceso ligero (Hilo):**

Carretero Pérez (2001) menciona que un **proceso ligero**, o **hilo** (thread), es un programa en ejecución (flujo de ejecución) que comparte la imagen de memoria y otras informaciones con otros procesos ligeros.

*Stallings menciona dos características:*

♦ La unidad que se activa se suele denominar hilo (thread), o proceso ligero

♦ La unidad de propiedad de recursos se suele denominar procesos o tarea

**Multi Hilo:** Menciona Stallings que el concepto de **MultiHilo** se re fiere a la capacidad de un SO de dar soporte a múltiples hilos de ejecución en un solo proceso.

**DOS**: posee un proceso por hilo

**Java**: Posee un proceso para múltiples hilos.

**Unix**: posee múltiples procesos, un hilo por proceso.

**Windows/GNU**: posee múltiples procesos y múltiples hilos por proceso.

**Composición de un proceso y sus hilos:**

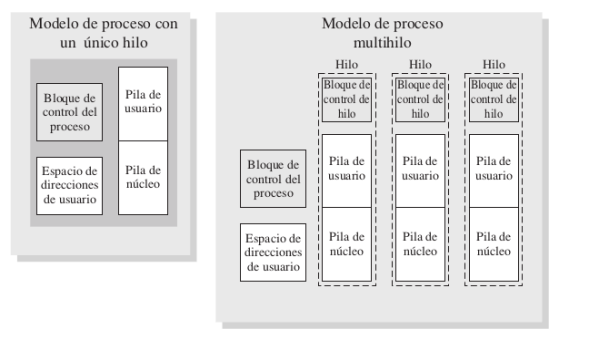
En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Se asocian con procesos los siguientes componentes:

♦ Un espacio de direcciones virtuales que soporta la imagen del proceso.

♦ Acceso protegido a procesadores, otros procesos (para comunicación entre

procesos), archivos y recursos de E/S (dispositivos y canales).

♦ Variables globales y estáticas.



**Dentro de un proceso puede haber uno o más hilos, cada uno con:**

♦ Un estado de ejecución por hilo (Ejecutando, Listo, etc.).

♦ Un contexto de hilo que se almacena cuando no está en ejecución, una forma

de ver a un hilo es como un contador de programa independiente dentro de

un proceso.

♦ Una pila de ejecución

♦ Por cada hilo, espaciado de almacenamiento para variables locales.

♦ Acceso a la memoria y recursos de su proceso, compartido con todos los

hilos de su mismo proceso

**Beneficios de los hilos:**

**1.** Lleva mucho menos tiempo crear un nuevo hilo en un proceso existente que

crear un proceso totalmente nuevo. Según estudios son 10 veces más

rápidos.

**2.** Lleva menos tiempo finalizar un hilo que un proceso.

**3.** Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.

**4.** Los hilos mejoran la eficiencia de la comunicación, ya que los hilos dentro de

un mismo proceso comparten memoria y archivos, se pueden comunicar

entre ellos sin necesidad de invocar al núcleo/kernel.

*“Según Stallings (2005):*

*A veces los hilos son también útiles en un solo procesador ya que ayudan a simplificar la estructura de programas que realizan varias funciones diferentes “*

**Ejemplos de uso de los hilos:**

**● Trabajo en primer plano y en segundo plano:** Por ejemplo, en un programa de hoja de cálculo, un hilo podría mostrar menús y leer la entrada de usuario, mientras otro hilo ejecuta los mandatos de usuario y actualiza la hoja de cálculo.

**● Procesamiento asíncrono:** Los elementos asíncronos de un programa se pueden implementar como hilos

● **Velocidad de ejecución:** Un proceso multihilo puede computar una serie de datos mientras que lee los siguientes de un dispositivo. En un sistema multiprocesador pueden estar ejecutando simultáneamente múltiples hilos de un mismo proceso.

● **Estructura modular de programas:** Los programas que realizan diversas tareas o que tienen varias fuentes y destinos de entrada y salida, se pueden diseñar e implementar más fácilmente usando hilos

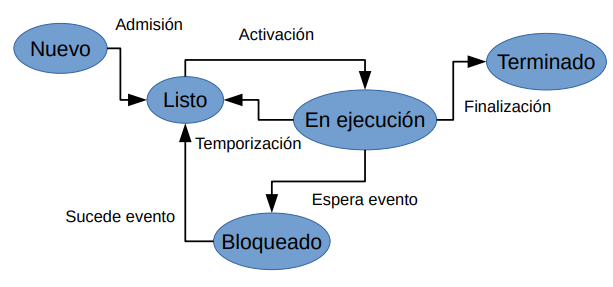
**Estados de los Hilos:**

● **Creación:** Cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Posteriormente, un hilo del proceso puede crear otro hilo dentro del mismo proceso, proporcionando un puntero a las instrucciones y los argumentos para el nuevo hilo. Al nuevo hilo se le proporciona su propio registro de contexto y espacio de pila y se coloca en la cola de Listos

● **Bloqueo:** Cuando un hilo necesita esperar por un evento se bloquea, almacenando los registros de usuario, contador de programa y punteros de pila. El procesador puede pasar a ejecutar otro hilo en estado Listo, dentro del mismo proceso o en otro diferente.

● **Desbloqueo:** Cuando sucede el evento por el que el hilo está bloqueado, el hilo se pasa a la cola de Listos.

● **Finalización:** Cuando se completa un hilo, se liberan su registro de contexto y pila.



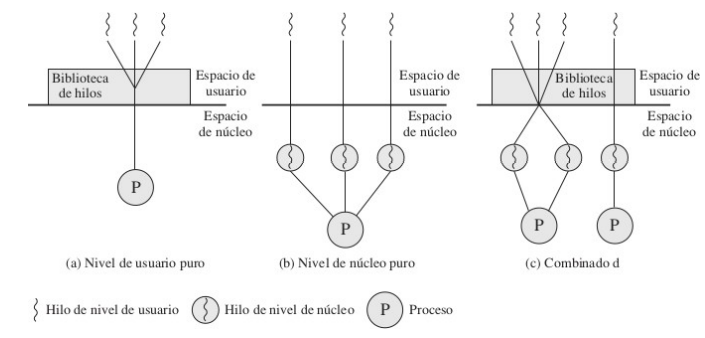
**Sincronización de hilos:**

Según Stallings (2005), todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos como, por ejemplo, los archivos abiertos. Cualquier alteración de un recurso por cualquiera de los hilos, afecta al entorno del resto de los hilos del mismo proceso. Por tanto, es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos. Por ejemplo, si dos hilos de modo simultáneo intentan añadir un elemento a una lista doblemente enlazada, se podría perder un elemento o la lista podría acabar malformada

**Categorías de implementación de hilos:**

Para Stallings (2005), existen dos amplias categorías de implementación de hilos:

**hilos de nivel de usuario (user-level threads, ULT)** e **hilos de nivel de núcleo (kernel-level threads, KLT)**. Los últimos son también conocidos en la literatura como hilos soportados por el núcleo ( kernel-supported threads) o procesos ligeros (lightweight processes)



**Hilos a nivel núcleo:**

● Son administrados por el núcleo (kernel)

● Utilizan la planificación a corto plazo (a nivel de hilo)

**Ventajas**

● Dos o más hilos se pueden ejecutar en procesadores diferentes.

● Si un hilo se bloquea, otro hilo del mismo proceso puede seguir su ejecución.

**Desventajas**

● Mayor sobrecarga (overhead) respecto de los hilos a nivel de usuario (ULT).

**Hilos a nivel usuario:**

● Son administrados por una biblioteca a nivel de usuario

● Utiliza doble planificación (a nivel de núcleo y a nivel de usuario)

**Ventajas**

● Menor sobrecarga (overhead) respecto de procesos y KLTs.

● Planificación propia.

● Portabilidad respecto del sistema operativo.

**Desventajas**

● Dos hilos de un mismo proceso, no se pueden ejecutar en distintos procesadores.

● Si un hilo se bloquea, se bloquea todo el proceso y sus hilos

Dentro del nivel de usuarios:

♦ Interceptar llamadas al sistema:

♦ Brinda más opciones de administración a la biblioteca de nivel

de usuario. Permite replanificar un cambio de contexto a nivel de

hilo.

♦ Queda a elección del programador.

En lugar de llamar a read(), se podría invocar a

read\_de\_la\_bilioteca().

♦ Jacketing(revestimiento): es una técnica que convierte una llamada

bloqueante en una NO bloqueante

*Ejemplos de implementación de hilos en GNU/Linux:*

♦ C

♦ NPTL

♦ LinuxThreads

Fin Unidad 2

**Concepto de Interbloqueo:**

Según Tanenbaum (2009), un conjunto de procesos se encuentra en un **interbloqueo** si cada proceso en el conjunto está esperando un evento que sólo puede ser ocasionado por otro proceso en el conjunto.

La definición de interbloqueo, según Stalling (2005), es el bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o se comunican entre sí.

**Definición de recurso:**

En el estudio de interbloqueos, diremos que los objetos otorgados son recursos. Y según Tanenbaum (2009), un recurso puede ser un dispositivo de hardware (por ejemplo, una unidad de disco) o una pieza de información (como un registro bloqueado en una base de datos)

**Tipos de recursos:**

**\*Apropiativo:** es uno que se puede quitar al proceso que lo posee sin

efectos dañinos. Ejemplo: memoria principal con intercambio

**\*No apropiativo:** es uno que no se puede quitar a su propietario actual sin

hacer que el cómputo falle. Ejemplo: grabación de un DVD

\***Reutilizable:** es aquél que sólo lo puede utilizar de forma segura un proceso

en cada momento y que no se destruye después de su uso. Ejemplos:

microprocesador, memoria principal y secundaria, entre otros

\***Consumible:** es aquél que puede crearse (producirse) y destruirse

(consumirse). Normalmente, no hay límite en el número de recursos consumibles de un determinado tipo. Ejemplo: señales

*Secuencia de eventos para utilizar un recurso:*

1. Solicitar el recurso
2. Utilizar el recurso
3. Liberar el recurso

**Condiciones para los interbloqueos de recursos:**

Coffman y otros colaboradores, afirman que deben existir cuatro condiciones para que exista un interbloqueo de recursos:

1. Condición de **exclusión mutua.**

*Cada recurso se asigna en un momento dado a sólo un proceso, o está disponible*

1. Condición de **contención/retención y espera.**

*Los procesos que actualmente contienen recursos que se les otorgaron antes pueden solicitar nuevos recursos*

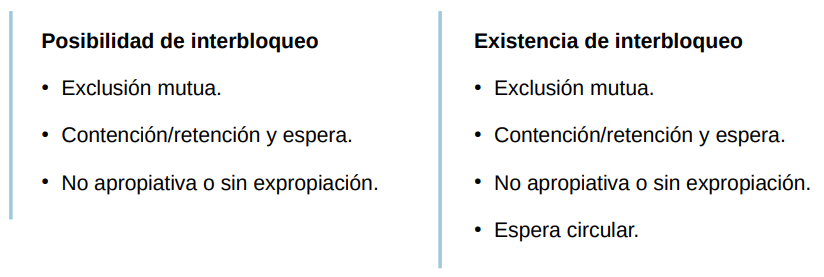
1. Condición **no apropiativa o sin expropiación**

*Los recursos otorgados previamente no se pueden quitar a un proceso por la fuerza. Deben ser liberados de manera explícita por el proceso que los contiene.*

1. Condición de **espera circular.**

*Debe haber una cadena circular de dos o más procesos, cada uno de los cuales espera un recurso contenido por el siguiente miembro de la cadena*

*Resumiendo:*



**Estrategias para lidiar con los interbloqueos:**

**\*Prevenir:** intenta que no ocurra una de las 4 condiciones necesarias y

suficiente para que exista un interbloqueo

**\*Predecir:** para evitar (Algoritmo del banquero, es una mejora de detectar y

recuperar

**\*Detectar y recuperar:**

**\*Algoritmo del avestruz:**

**Prevención:** Garantiza que no ocurran los interbloqueos

No permite que se produzca alguna de las siguientes cuatro condiciones necesarias y suficientes: (Exclusión mutua, retención y espera, sin expropiación y espera circular).

**Exclusión mutua:** Todos los recursos pueden ser utilizados por todos los procesos del SO. En la realidad, no puede eliminarse la presente condición, nos da más problemas que soluciones.

**Contención/retención y espera:** En *una primera alternativa*, los procesos pueden solicitar todos los recursos juntos.

Una *segunda alternativa*, los procesos solicitan un recurso a la vez, lo utilizan y lo liberan.

**Sin expropiación:** *Primer alternativa:* Un proceso con recursos asignados, solicita un recurso que se encuentra utilizado por otro proceso, éste debe liberar sus recursos asignados.

*Segunda alternativa:* Un proceso P1 solicita un recurso que se encuentra asignado a otro proceso P2 que se encuentra esperando por más recursos. El recurso asignado al proceso P2 puede asignarse al proceso P1.

**Espera circular:** Los recursos deben tener un orden asignado. Y los recursos solo pueden solicitarse en orden creciente.

**Algoritmo del banquero:** (Algoritmo que puede evitar los interbloqueos)

El presente algoritmo se encuentra asociado al funcionamiento de un banco, donde se debe asegurar la disponibilidad de dinero, en el caso de que todos los clientes del banco requieran retirar el máximo de sus cuentas

*Cuenta con las siguientes estructuras de datos:*

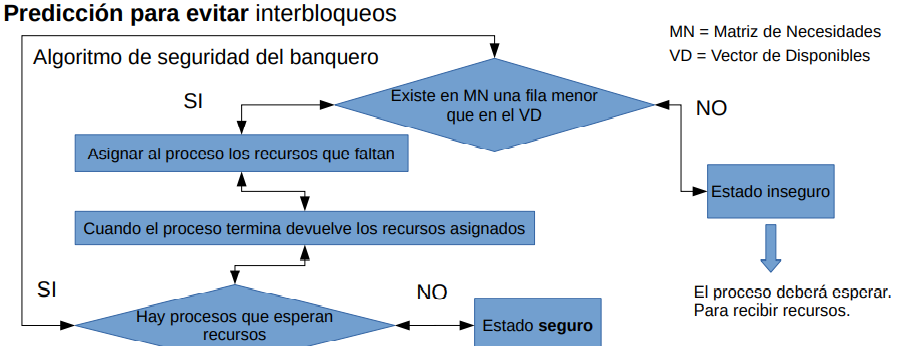
- Vector de recursos existentes.

- Vector de recursos disponibles.

- Matriz de solicitados.

- Matriz de asignados.

- Matriz de necesidades.



**Detección y recuperación:**

-Si puede ocurrir un interbloqueo en el sistema.

-No existen restricciones para asignar recursos a los procesos.

- Periódicamente, en el SO se ejecuta un algoritmo que detecta la presencia de un interbloqueo.

- Se ve una actitud más reactiva que proactiva, en contraposición a la estrategia anterior.

*Estrategias para recuperarse de un interbloqueo:*

1. Abortar todos los procesos involucrados en el interbloqueo.
2. Retroceder cada proceso en el interbloqueo a algún punto de control (checkpoint) previamente definido y arrancar todos los procesos.
3. Abortar sucesivamente los procesos en el interbloqueo hasta que este deje de existir.
4. Expropiar sucesivamente los recursos hasta que el interbloqueo deje de existir.

*Criterios para las estrategias 3 y 4:*

*●* menor cantidad de tiempo de procesador consumida hasta ahora

● menor cantidad de salida producida hasta ahora

● mayor tiempo restante estimado

● menor número total de recursos asignados hasta ahora

● menor prioridad

**Algoritmo del avestruz:** Básicamente desconoce completamente la existencia del problema.

Se utiliza dada la baja frecuencia de ocurrencia de interbloqueos en el sistema (o cuando es posible reiniciar el sistema sin problema alguno).

Sistemas operativos como GNU/Linux y Windows lo implementan