Módulo 3

PLANIFICACIÓN DE PROCESOS Y PROCESADORES

CONTENIDO:
- Conceptos sobre planificación de trabajos y de monoprocesador y multiprocesadores.
- Planificación de trabajos en Tiempo Real.
- Distintos algoritmos de planificación.
- Entender como planifica los procesos el S.O. UNIX.

OBJETIVO DEL MÓDULO: Describir los distintos métodos y políticas utilizadas para planificar la ejecución de trabajos y la administración de procesadores empleados en los Sistemas Operativos.

OBJETIVOS DEL APRENDIZAJE: Después de estudiar este módulo, el alumno deberá estar en condiciones de:

- Conocer y explicar las razones de la existencia de los diversos planificadores de un SO
- Conocer y comprender las políticas y los mecanismos de los algoritmos de planificación.
- Estudiar distintos algoritmos de administración de procesadores.
- Entender como planifica los procesos el sistema operativo.
- Conocer la terminología y sus significados utilizados en éste módulo.

Meta del Módulo 3

El presente Módulo tiene por meta profundizar los conceptos sobre un tema fundamental en los sistemas operativos modernos, como es la planificación de procesos y procesadores. Además permitirá comprender la real importancia que la planificación tiene sobre el funcionamiento global de sistemas de cualquier tamaño, permitiendo la comprensión de distintos fenómenos que se presentan habitualmente. Por último busca introducir un cambio en la forma de trabajar ya que permite entrar en conocimiento con técnicas que tienden a optimizar la ejecución de cualquier tarea. Comenzaremos dando los conceptos sobre monoprocesadores, luego pasaremos a multiprocesadores y por último la planificación en Tiempo Real.

3.1. Introducción al problema de la Planificación: Planificación de Monoprocesadores.

A pesar que la planificación es una de las funciones fundamentales del sistema operativo durante un largo periodo los más populares no contaban con algoritmos sofisticados para optimizar la ejecución, sino que era secuencial (Batch). Esto producía un desaprovechamiento muy importante de las capacidades del procesador ya que la ejecución de un proceso alterna entre dos estados de ejecución: uno de CPU y otro de Entrada / Salida, por lo que mientras se trabajaba con un dispositivo el procesador se encontraba inactivo.

Los sistemas operativos evolucionaron hacia lo que se denominó multiprogramación. Esta nueva forma de trabajo trajo consigo nuevos problemas, haciendo necesaria la aparición de algún método que permitiera organizar la forma en que se debían ejecutar los procesos, surgiendo así la planificación.

Entonces podríamos definirla como un **conjunto de políticas y mecanismos incorporados al sistema operativo, a través de un módulo denominado planificador**, que debe decidir cual de los procesos en condiciones de ser ejecutado conviene ser despachado primero y qué orden de ejecución debe seguirse.

La planificación y administración de trabajos (un trabajo es un programa y sus datos) y del procesador, se ocupa de la gestión de la ejecución; para esto, el S.O. usa diferentes políticas y mecanismos.

La Planificación es el conjunto de políticas y mecanismos incorporados al S.O. que gobiernan el **orden** en que serán ejecutados los trabajos, cuyo ***objetivo*** principal es el máximo aprovechamiento del Sistema, lo que implica proveer un buen servicio a los procesos existentes en un momento dado.

De acuerdo a lo explicado en el módulo anterior los programas se cargan en Memoria Central en donde, el Sistema Operativo crea los procesos que luego son ejecutados.

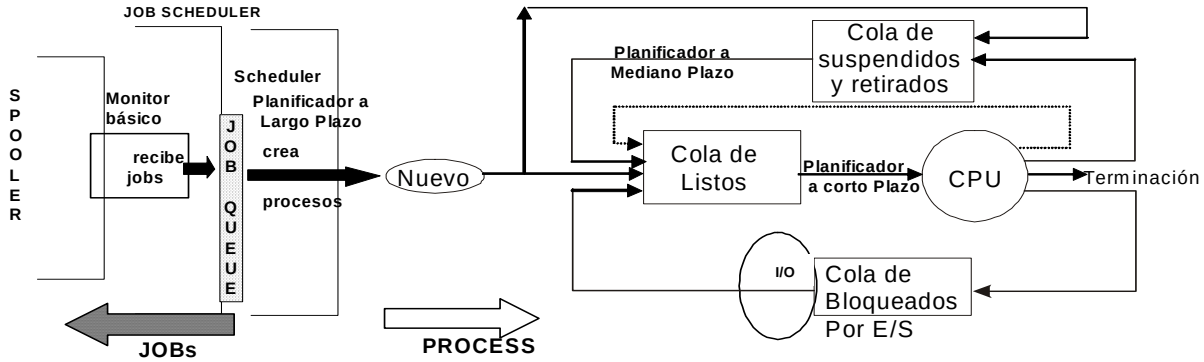


Figura 3.01 Instancias de un Job y de un Proceso.

La figura 3.01 muestra el recorrido de un proceso desde que el trabajo es sometido inicialmente al sistema (Spooler) hasta que es finalizado. Este recorrido se conoce como ciclo de vida de un proceso que comienza con su creación, su estado de listo para ejecutar dentro de la cola Ready, luego pasa al estado de ejecución y de ahí puede pasar a la cola de Bloqueados (Waiting) cuando requiere un servicio de E / S o volver a la cola de listos o completarse.

Se plantean dos problemas:

- Si la carga de procesos que ejecutan concurrentemente es alta, los tiempos de ejecución aumentan debido al multiplexado de los recursos, lo que se conoce como ***thrashing***. (**Thrashing** -azotado del procesador - o sea, la pérdida de desempeño debido al trabajo innecesario causado al procesador por tener que multiplexar los recursos).
- En caso de política secuencial (Batch), los tiempos de espera de los trabajos son grandes por lo que se producen tiempos ociosos dentro del procesamiento.

Entre varios procesos en condiciones de ser ejecutados, el sistema operativo debe decidir cual conviene despachar primero y qué orden de ejecución debe seguirse.

El módulo del sistema operativo encargado de esta tarea se denomina ***planificador*** (**scheduler**) o ***administrador del procesador*** y el algoritmo particular utilizado se denomina ***algoritmo de planificación***.

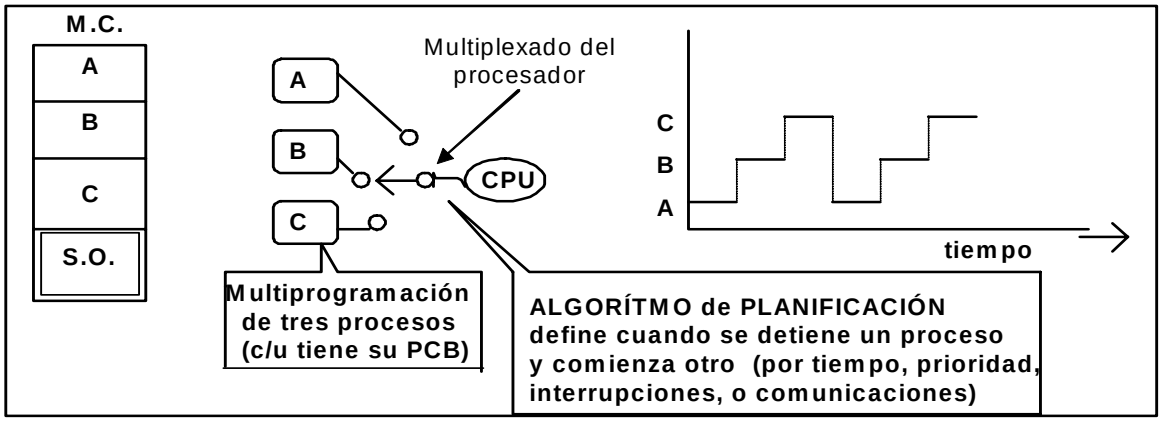


Fig. 3.02 El procesamiento Multiprogramado con 3 programas

3.2. Niveles de Planificación

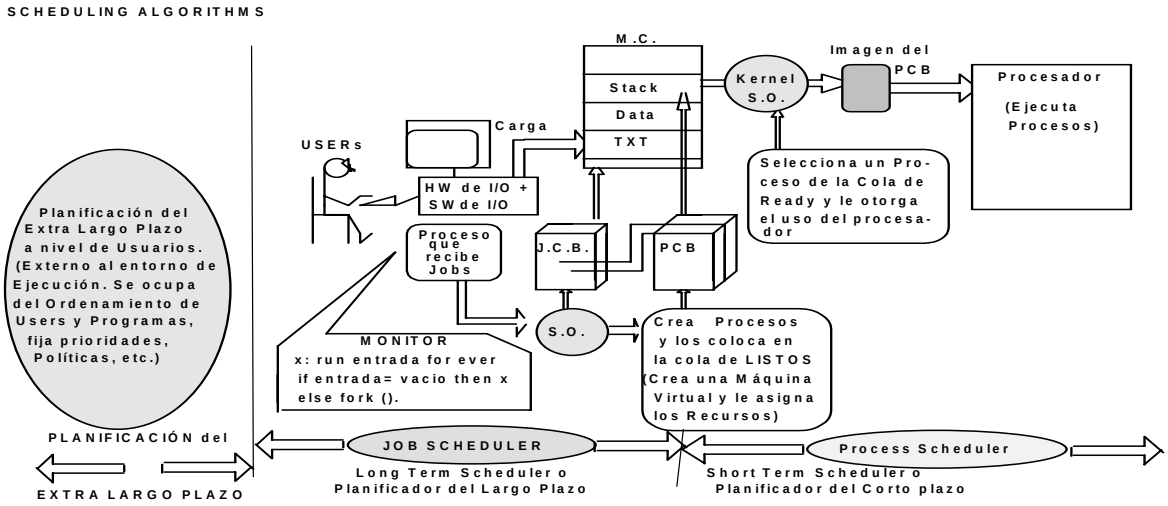


Fig. 3.03 Niveles de planificación

La planificación se hace en cuatro instantes de tiempo. De estas cuatro, una no la realiza el S.O. sino es externa al procesamiento, pero tiene una influencia enorme sobre la definición del procesamiento, dado que el S.O. queda determinado por las decisiones que se toman en este nivel. A esta instancia le dimos el nombre de **Extra Largo Plazo** por ser en la escala de tiempo del ser humano.

En la administración del procesador podemos distinguir tres niveles de planificación de acuerdo a la escala de tiempo en que se realiza la misma. El Largo plazo en Segundos, Mediano plazo en Milisegundos y el Corto plazo en nanosegundos o microsegundos.

3.2.1.Planificación en el Extra Largo Plazo:

Consiste un una planificación externa que se hace en el centro de cómputos y esta estrechamente ligada a las políticas de funcionamiento del sistema. Es una planificación que se hace en concordancia con la política empresaria en que define el funcionamiento del centro de cómputos y se plantean las prioridades de usuarios o procesos, se organizan los programas que se van a ejecutar fuera de línea y ya encolados se los lleva a ejecución. Se realiza en el ámbito de usuarios y define las políticas de funcionamiento del sistema.

A través de procedimientos escritos se fijan las reglas que se aplicaran a los usuarios relativos al uso, seguridad, accesos, prioridades, etc.

Decíamos que es externo al sistema de cómputos y fundamentalmente político ya que determina la importancia relativa de los usuarios. Depende de la organización del centro de cómputos y para regularlo se deben crear ***Procedimientos escritos*** que fijen claramente las ***Reglas*** de uso, seguridad,

accesos, prioridades, etc. de cada usuario, además de la modalidad de procesamiento, de la operación, de la política de backup, etc.. En realidad todas las actividades deberían estar reglamentadas.

Esta planificación busca satisfacer cuatro objetivos desde el punto de vista de los usuarios:

- Mayor velocidad de respuesta en sus trabajos con lo que disminuye el tiempo de espera de los usuarios.
- Existencia y disponibilidad de recursos que necesitan para ejecutar sus trabajos.
- Importancia de sus tareas.
- Seguridad de que sus trabajos sean completados correctamente.

La implementación de la política de planificación obviamente debe garantizar el cumplimiento de estos objetivos, permitiendo así que todo funcione según lo establecido

Es de destacar que ésta planificación es determinante para que un centro de cómputos funcione correctamente o no.

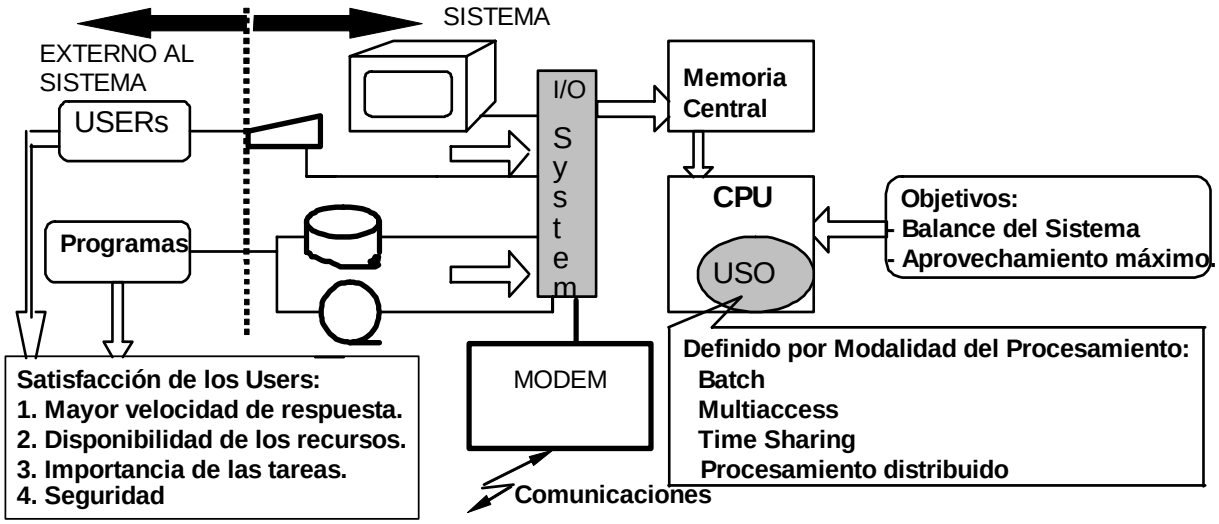


Fig. 3.04 Planteo de los problemas en el procesamiento

Es obvio que si no se satisfacen alguno de estos objetivos, los usuarios se quejarán manifestando sus disconformidades y en un caso extremo dejarán de usar el sistema.

En la Fig. 3.04 se observa en detalle los problemas planteados. Sobre todo si el Sistema (Hardware + Software + organización + ambiente) no responde adecuadamente a los cuatros objetivos planteado. Entonces, los usuarios y sus programas, expresarán sus disconformidades, dando los detalles del problema por el cual no funciona el sistema. En particular es la responsabilidad del profesional de sistemas brindar un adecuado servicio de procesamiento de datos como también ocuparse del ***orgware*** y del ***peopleware*** para que todo funcione dentro de lo establecido.

3.2.2. Planificación a largo plazo

Está a cargo del módulo del S.O. llamado planificador de trabajos, Job scheduler o long-term scheduler. El trabajo es recibido por el S.O. a través del monitor (o un demonio) que es un software que se ocupa de cargar el trabajo en M.C. (memoria central).

Long Term Scheduler (Planificador de Trabajos): decide cual será el próximo trabajo que se ejecutará, para lo cual carga el Programa y sus datos y crea los procesos. Es importante en Sistemas Batch donde decide sobre la base de las necesidades de los recursos que requieren los procesos, cuál de ellos logrará mantener el procesador.

Después de haber sido recibido el trabajo se prepara y se crean Procesos con sus respectivos Bloques de Control (PCB o vector de estado) para poder ejecutarlos. Si los recursos que solicita estuvieran disponibles, se le asignan y se lo ingresa a una ***Cola de Listos para ejecutar (Ready Queue***) o cola de ejecución como se llama en UNIX. Este nivel de planificación es realizado por el ***Planificador de trabajos o Job Scheduler*** *(algunos autores lo designan Long term scheduler, macroscheduler, high level scheduler o simplemente scheduler).*

El objetivo del ordenamiento de los procesos en el uso del procesador es minimizar el costo total de los servicios de cómputo y del tiempo de espera de los usuarios brindándoles la máxima satisfacción.

Son dos Problemas distintos:

- Minimizar costo y
- Minimizar los tiempos de espera.

Lo que implica tomar decisiones en distintos niveles. Lo explicado en el extra largo plazo (Fuera del S.O. en el ámbito organizacional y político), tiene una gran influencia sobre el funcionamiento del sistema. Dado que el sistema va a responder de acuerdo a esas políticas y como se planifica en el largo plazo.

El largo Plazo lo definimos dentro del S.O. como Long Term Scheduler o Planificador de trabajos (algunos Sistemas incluyen un Middle Term Scheduler) y en el corto Plazo (Short term scheduler o planificador de procesos).

El objetivo del largo plazo es proporcionar una mezcla equilibrada de trabajos, tales como limitados por el procesador y limitados por Entrada / Salida. Esto se conoce como ***Balance de carga del sistema***. En cierto modo actúa como un regulador de admisión de primer nivel para mantener la utilización de recursos al nivel deseado sin sobrecargar al sistema. Entonces debe seleccionar con cuidado que tipos de trabajos entran al sistema para mantener un adecuado balance de carga de procesos que hacen uso de Entrada/Salida intensivo (**I/O bound** - limitado por Entrada / Salida) o uso de procesador intensivo (**CPU bound** - Limitado por el uso de procesador).

Observar que el principal OBJETIVO del sistema es minimizar el costo debido al tiempo perdido a factores tales como:

- Intervenciones del Operador.
- Periféricos lentos,
- Multiplexado de recursos.

La figura 3.05 ejemplifica esto e indica la instancia en que se aplica el algoritmo de planificación.

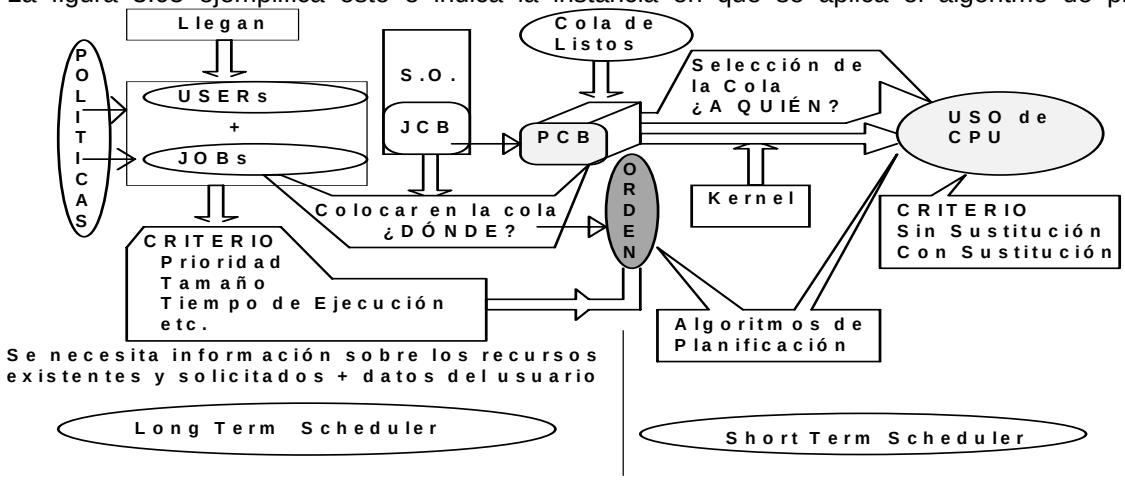


Fig. 3.05 Planteo de los problemas de ordenamiento y sus criterios

Para ello existen diferentes filosofías en el procesamiento de un trabajo. Todas ellas responden a ciertos criterios de planificación que se vuelcan en los respectivos algoritmos de planificación. Esto se conoce como la modalidad de ejecución o procesamiento. Los más importantes son:

* BATCH apunta estrictamente al exhaustivo uso del procesador en detrimento del Usuario. Sus principales características son:
  1. El procesador es monoprogramado.
  2. No existe diferencia entre trabajo y proceso.
  3. El Scheduler elige el trabajo, crea el proceso y lo ejecuta.
  4. Prácticamente hay un solo nivel de planificación.
* INTERACTIVO: apunta al servicio del usuario en detrimento de la performance del procesador. Es multiprogramado pues se multiplexa el procesador entre varios Programas.
* MULTIPROCESADO es un ambiente en el que existen varios procesadores para servir a los procesos en ejecución.
* PROCESAMIENTO DISTRIBUIDO O EN RED: Es una forma de procesamiento en que se le presenta al usuario una máquina virtual y en que el procesamiento se realiza en distintas máquinas diseminadas geográficamente y que están conectadas por una red.

Esta modalidad de procesamiento lo estudiaremos en detalle en los módulos sobre Procesamiento Distribuido.

El scheduler es un administrador que se encarga de organizar la ejecución con un adecuado planeamiento de recursos para que el trabajo ejecute ordenadamente y eficientemente según se determino la modalidad de procesamiento. Se lo puede considerar como un ***capataz general*** que controla que todo funcione correctamente. Generalmente, en modo batch se usa un lenguaje particular para manejar los trabajos llamado ***Job Control Language*** (J.C.L.).

El Scheduler ejecuta con poca frecuencia; sólo cuando se necesita crear un proceso nuevo en el Sistema por la entrada de un trabajo, cuando termina un trabajo mediante un proceso de finalización, o ingresa un usuario al sistema, por lo que tiene prioridad máxima para ejecutar. Es el responsable de controlar el nivel de multiprogramación del sistema y generar el orden a seguir en la ejecución de los procesos.

El monitor reside en el kernel del S.O. y tiene las siguientes funciones:

a) Recibir los trabajos, generar una Tabla (Job Control Block) y crear procesos como nuevos.
b) Recibir a los Usuarios cuando llegan. Esto se conoce como ***login***, por lo que en sistemas de tiempo compartido limita la cantidad de usuarios que pueden acceder al sistema.
c) Tomar los procesos nuevos y pasarlos a la cola de listos (ready queue) si hay disponibilidad en memoria o enviarlos al disco si no hay.

* **Funciones:**

1. Tener actualizados los bloques de control de los trabajos (JCB),
2. Identificar usuarios y trabajos ordenados por los usuarios y verificar los permisos.
3. Tener actualizadas las tablas de recursos disponibles.
4. Ordenar la cola de Listos.

Es muy poco usado en sistemas de tiempo compartido, en su lugar, se introduce al **medium-term scheduler** que mantiene un equilibrio entre los procesos activos e inactivos. En la Fig. 3.06 se observa un esquema de este tipo de planificación.

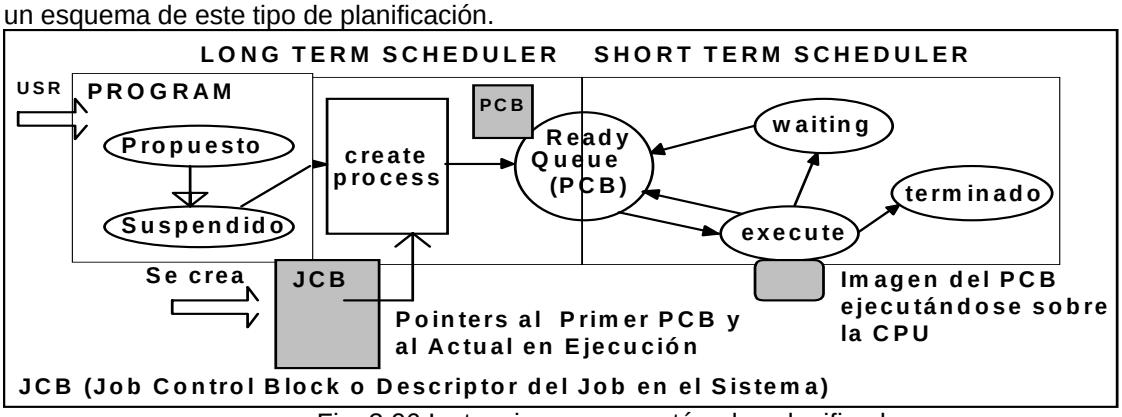


Fig. 3.06 Instancias en que actúan los planificadores.

* Las Tareas del Planificador de Trabajos son:

- Mantener un registro de estado de todos los trabajos en el Sistema (JCB).
- Establecer una estrategia para el pasaje entre las colas de suspendidos y la de listos (Ready).
- Asignar recursos (memoria central, dispositivos, procesadores, etc.) a cada trabajo.
- Pedir (recuperar) los recursos cuando los trabajos se han completado.
- Detectar y prevenir los conflictos de Abrazo Mortal o deadlock.

(Deadlocks lo estudiaremos en el próximo módulo).

- Dar entrada a nuevos trabajos (en Batch significa ponerlos en el pool de tareas en memoria secundaria, en multiaccess significa que los procesos son creados cuando el usuario accede al Sistema (por lo que se limita el número o cantidad de Usuarios o trabajos).
- Asignar prioridades a los procesos. Esto genera el ***orden de ejecución*** y viene determinado básicamente por el orden de procesos en la Cola (Ready Queue), o sea, el orden en que el ***dispatcher*** los seleccionará de esta cola para ponerlos en ejecución (generalmente el primero de la cola).

El Scheduler implementa el orden de la cola.

Job Id.
Estado del Job
Pointer a la posición de colas de jobs
Pointer al primer proceso del Job
Pointer al Proceso actual (Activo)
Prioridad asociada
Tiempos estimados de ejecución
Recursos,
etc.

Fig. 3.07. Modelo de una estructura de datos para controlar los Trabajos que ingresan al Sistema.

- Implementar las políticas de asignación de recursos (razón por la que se le otorga la máxima prioridad en el sistema para que el dispatcher lo seleccione primero si está libre el procesador y se ejecuta cuando:
  i. Se pide o libera un recurso[1]
  ii. Cuando termina un proceso.
  iii. Cuando llega un nuevo trabajo al pool de procesos (lo ubica en la Ready Queue)
  iv. Cuando llega un nuevo usuario al sistema.

Cuando el Scheduler termina de ejecutar se bloquea a sí mismo detrás de un semáforo y lo activan los cuatro eventos anteriores, o por medio de una interrupción.

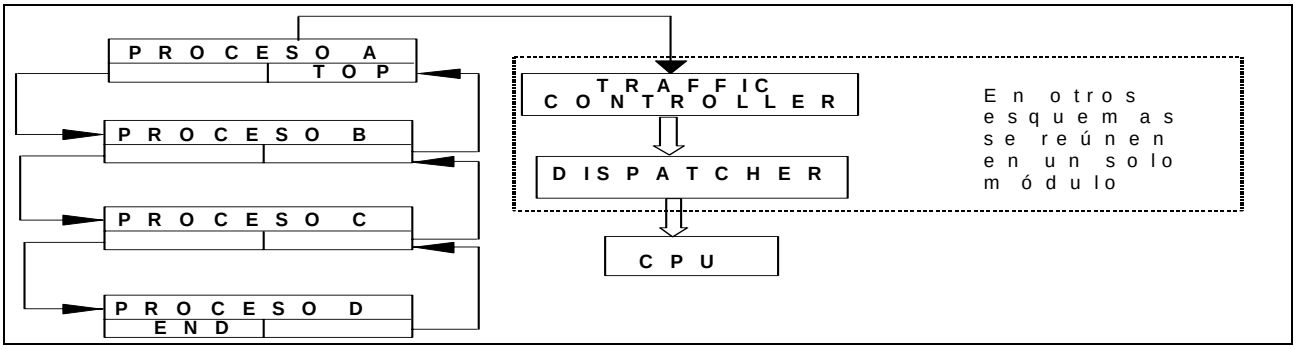
El Scheduler, entonces, define cuándo introducir procesos nuevos y el orden en seguir la ejecución de los procesos por lo que es el ***padre*** de todos los procesos.

La Cola de Listos (Ready Queue) contiene punteros a los PCB (Process Control Block) de los procesos listos o preparados para ser ejecutados como se indica en la Fig. 3.08. Su organización depende del algoritmo de planificación empleado para introducir los Trabajos.

El orden en el cual son ingresados inicialmente los trabajos lo decide el Planificador de Trabajos (long term scheduler o simplemente Scheduler) como se observa en la Fig. 3.06.

Al entrar un Programa al Sistema, el Scheduler crea una estructura de datos (JCB Job Control Block o Descriptor del Trabajo) para controlar su ejecución.
La información que contiene es proporcionada por los usuarios, comandos de control y por el sistema. (varía de S.O. en S.O.) . Un ejemplo de contenido se observa en la Fig. 3.07.

En la Fig. 3.07 se presenta la estructura de datos llamada ***Job Control Block (JCB)*** que se crea cuando el trabajo es aceptado por el sistema operativo. Su principal objetivo es permitir controlar la ejecución del trabajo y darle existencia en el sistema.



[1] El pedido o la liberación de recursos se realizan mediante llamadas del sistema: ***pedir_recursos*(recurso, resultado) y *liberar_recursos*(recurso)** Estas primitivas se ejecutan con mucha frecuencia lo que produce un ***overhead*** muy grande si están incorporados en el Scheduler por lo que conviene incorporarlas en el Kernel y así evitar la invocación del Scheduler cada vez que se requiera asignar recursos.

Overhead**:** Es el tiempo que el S.O. emplea la CPU para el procesamiento propio (o sea, el tiempo invertido por el procesador en ejecutar rutinas del S.O.)

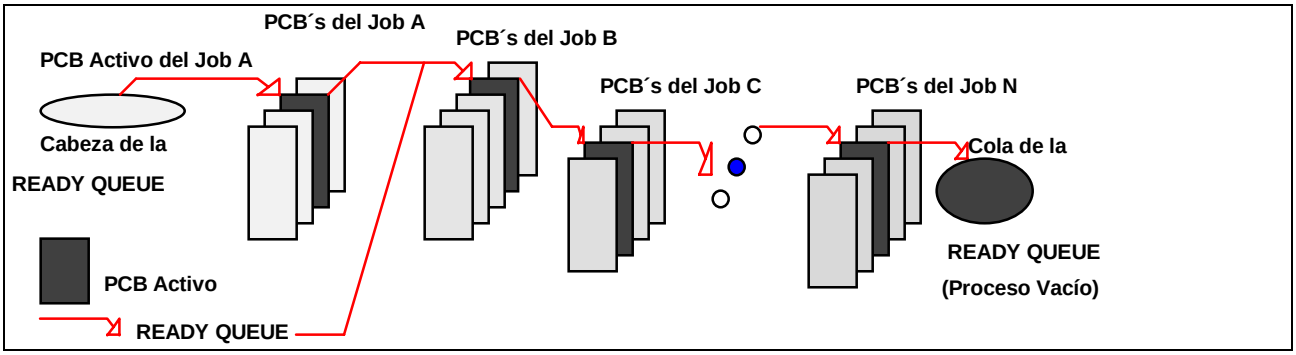


Fig. 3.08 Componentes que actúan en el corto plazo.

3.2.3. Planificación a mediano plazo

Está a cargo del módulo llamado medium-term scheduler. Este tipo de planificador se encuentra solo en algunos sistemas especialmente en los de tiempo compartido, ya que permite mantener un equilibrio entre los procesos activos e inactivos.

Middle Term Scheduler (también Medium term scheduler o Planificador de Swapping): es el que decide sacar de memoria central y llevar al disco (swap-out) a aquellos procesos inactivos o a los activos cuyos estados sean bloqueado momentáneamente o temporalmente o los suspendidos y luego, cuando desaparezcan las causas de su bloqueos, traerlos nuevamente a Memoria (swap-in) para continuar su ejecución

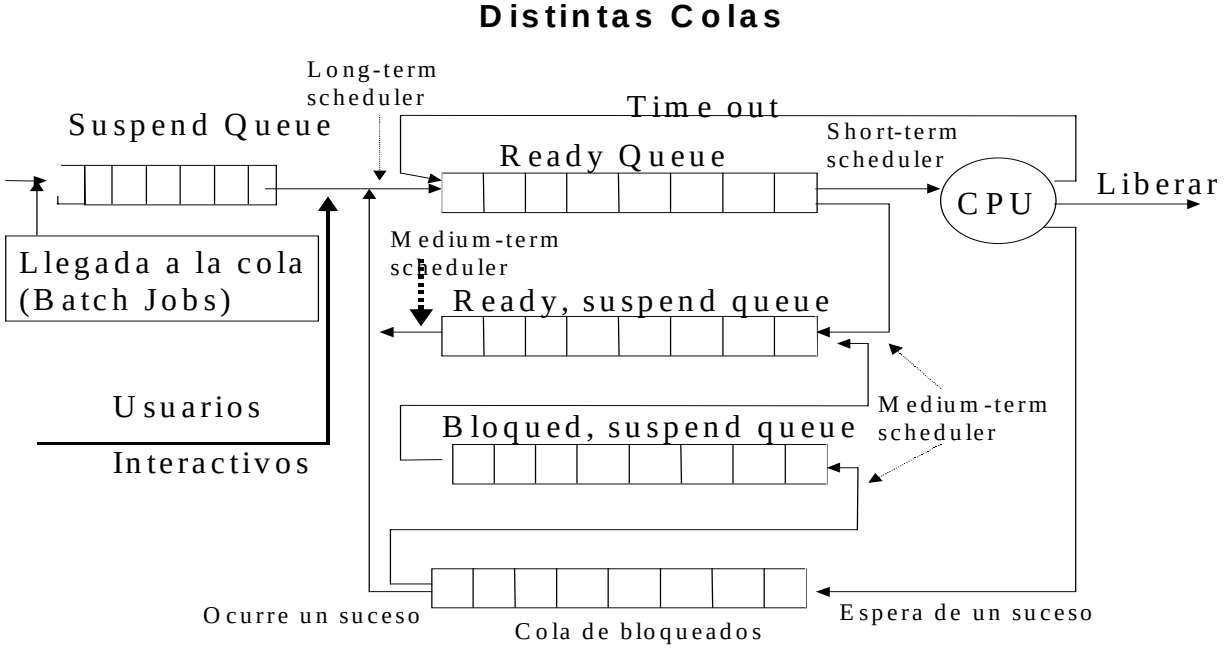


Fig.3.09. Colas y estados de procesos

Después de ejecutarse durante un tiempo determinado un proceso puede resultar suspendido al efectuar un pedido de servicio de Entrada / Salida o al emitir una llamada al sistema. Como los procesos suspendidos no pueden terminar de ejecutarse hasta que se extinga la condición de suspensión, puede ser importante quitarlos de la memoria central y guardarlos en almacenamientos secundarios, permitiendo así la entrada de nuevos procesos a la cola de listos. Este procedimiento se denomina ***swapping***[2]. Una vez desaparecida la condición de suspensión, el planificador a medio plazo intenta colocar el proceso suspendido nuevamente en memoria y dejarlo preparado para continuar con la ejecución.

Este planificador puede ser invocado cuando quede espacio libre de memoria por efecto de la terminación de un proceso o cuando el suministro de procesos caiga por debajo de un límite especificado.

[2] Swapping es el intercambio entre dos niveles de memoria. En este caso Memoria Central y Memoria secundaria (disco).

En algunos casos suplanta al long-term scheduler, y otros lo complementa: Por ejemplo en sistemas de tiempo compartido el long-term scheduler puede admitir más usuarios de los que pueden caber realmente en memoria. Sin embargo, como los trabajos de estos sistemas están caracterizados por ciclos de actividad y ciclos de ociosidad, mientras el usuario piensa algunos procesos pueden ser almacenados y al recibir respuesta vueltos a poner en la cola de listos.

Este tipo de planificación solo es usado en sistemas con mucha carga de procesos, ya que el procedimiento de swapping produce mucho overhead, haciendo bajar considerablemente el desempeño general.

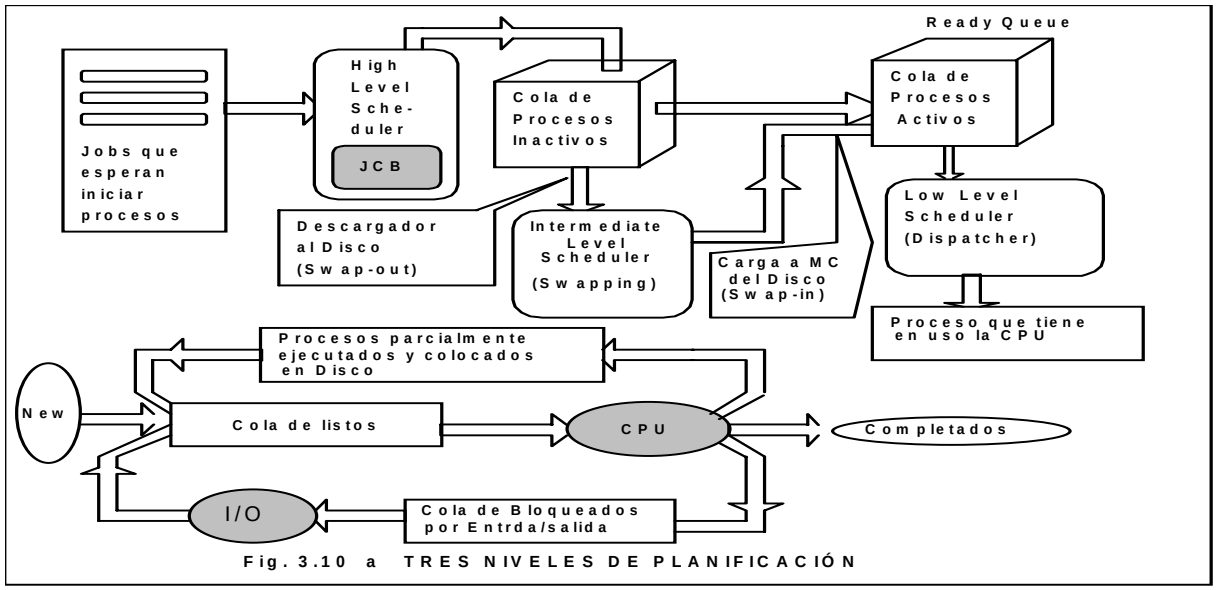


Fig. 3.10 a TRES NIVELES DE PLANIFICACIÓN

En la figura 3.10a se presenta un esquema que responde a la Planificación en tres niveles con una representación en tres niveles, Long term scheduler, middle term scheduler mas conocido como swaper y short term scheduler

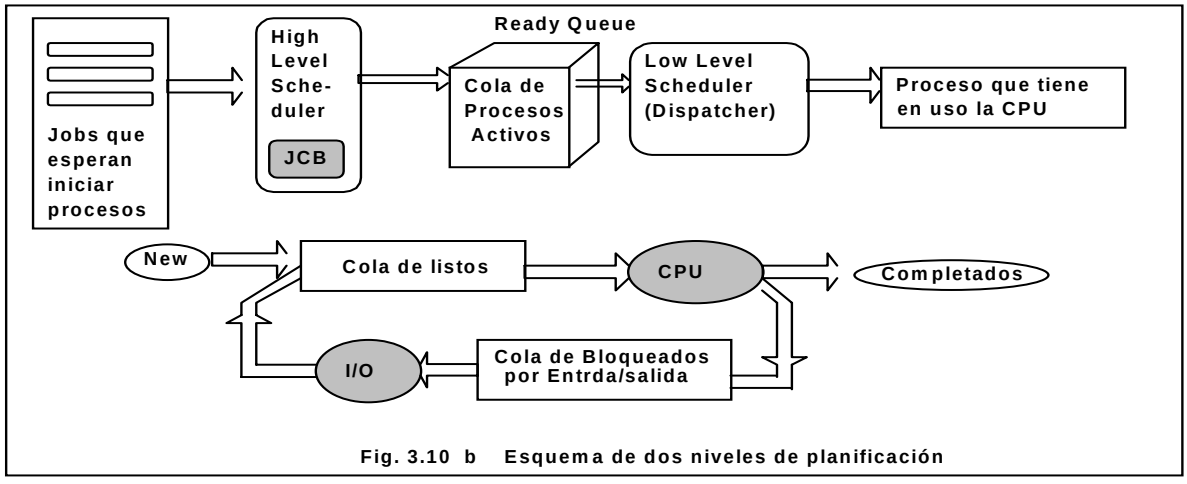


Fig. 3.10 b Esquema de dos niveles de planificación

Hay Sistemas Operativos que no utilizan este nivel intermedio, entonces tienen solo dos planificadores, el de largo plazo y el de corto plazo como se ve en la figura 3.10.b .

3.2.4. Planificación a corto plazo

Está a cargo del módulo llamado short-term scheduler, process scheduler o low scheduler.

Su función principal es la de asignar el procesador entre el conjunto de procesos preparados y listos para ejecutar residentes en memoria, teniendo por objetivo maximizar el rendimiento del sistema de acuerdo con el conjunto de criterios elegidos.

Planificador de Procesos (Short Term Scheduler): Es el responsable de decidir quién, cuándo, cómo y por cuánto tiempo recibe procesador un proceso que esta preparado en la Ready Queue para ejecutar, además en S.O. con esquemas expropiativos se ocupa de quitar el recurso procesador al proceso que esta ejecutando. También verifica las interrupciones y las trata.

La decisión es muy rápida dado que se toma el siguiente proceso de la ***Cola de listos para ejecutar*** y se lo pone a ejecutar (***Running***). Esta tarea es realizada por el ***Planificador de procesos.*** Los recursos a esta altura ya deben estar todos disponibles para este trabajo.

El planificador a corto plazo es invocado cada vez que un suceso (interno o externo) hace que se modifique el estado global del sistema. Por ejemplo:

- Tics de reloj (interrupciones basadas en el tiempo).
- Interrupciones por comienzo y terminaciones de Entrada / Salida.
- La mayoría de las llamadas al S.O..
- El envió y recepción de señales.
- La activación de programas interactivos.

Dado que cualquiera de estos cambios podría hacer que un proceso en ejecución sea suspendido, que a uno o más procesos suspendidos pasen a preparados, o que se agreguen nuevos proceso a la cola de listo, el planificador a corto plazo tendría que ser ejecutado para determinar si tales cambios significativos han ocurrido realmente y, si verdaderamente es así, seleccionar el siguiente proceso a ejecutar. Por tanto el low scheduler debe ser rápido y con poca carga para el procesador para que se mantenga el rendimiento, ya que se le debe sumar además el tiempo que toma en realizar el cambio de contexto (dispatch latency o context switch).

Es un administrador de muy corto plazo. En esta etapa se necesitan los recursos ya disponibles preparados por el planificador anterior (job scheduler). Se ocupa de seleccionar a uno de los procesos de la Ready Queue y le entrega el procesador para su ejecución.

En un Mainframe[3] está compuesto básicamente por dos módulos residentes en el Kernel:

1. **Traffic Controller:** Se ocupa del manejo de la cola de procesos listos. Es quien arma la cola de procesos listos o preparados para ejecutar para lo cual usa un algoritmo de planificación.
2. **Switcher o dispatcher**: otorga el uso del procesador al primer proceso de la cola y lo pone en ejecución.

La Ready Queue se forma con todos los procesos activos de todos los trabajos que están esperando por procesador. La cola está encabezada por el proceso que tiene la mas alta prioridad y es finalizada por el proceso vacío o nulo. Esto se observa en la Fig. 3.08.

Algunos S.O. unifican los dos módulos en uno solo que cumple con la función del cambio de contexto (context switch) estudiado en el módulo 2., por lo que debe observarse como es brindado el correspondiente servicio. Por ejemplo en UNÍX se tiene el switcher.

Otros, Planifican los procesos de acuerdo a una política y luego los procesos crean nuevos que se ejecutan en el mismo nivel de prioridades. Ejemplo de esto son los Procesos livianos que no producen grandes movimiento de datos en el context switch.

El proceso nulo o vacío

Un problema que debe resolver un Sistema Operativo multitarea es, que debería hacer el sistema cuando no hay nada que ejecutar. Por ejemplo cuando la cola de Listos se encuentra vacía.

En principio, esta situación requiere que el sistema espere hasta que un proceso haya llegado a la cola de Listos, lo seleccione y luego le ceda el control del procesador a ese proceso.

Una implementación sencilla consiste en que el planificador compruebe constantemente la cola de listos para ver si su contenido es no vacío, hasta que llegue un proceso y se realice el cambio de contexto. Esta estrategia no resulta efectiva en muchos procesadores donde el planificador se ejecuta con una prioridad mayor a todos los procesos usuarios, con el fin de quedar protegido contra expropiaciones potencialmente dañinas.

Mientras se ejecuta el sistema operativo en un ciclo de alta prioridad, el S.O. nunca puede ser capaz de detectar el suceso de la llegada de un proceso usuario a la cola (pues el proceso del usuario es de menor prioridad).

Este problema es resuelto en muchos sistemas operativos con el proceso NULO que es creado por el sistema en el momento del arranque. El proceso NULO nunca termina, no tiene E/S (Ej.: ciclo while,

[3] Mainframe es un macrocomputador o sea un computador grande o un sistema grande.

True do nada) y tiene la prioridad más baja en el sistema. En consecuencia la cola de listos nunca está vacía, además la ejecución del planificador puede hacerse más rápida al eliminar la necesidad de comprobar si la cola de listos está vacía o no. Algunas de las tareas que se le pueden dar al proceso NULO, por ejemplo, es realizar estadísticas de uso de procesador, o asistencia de vigilancia de la integridad del sistema, etc.

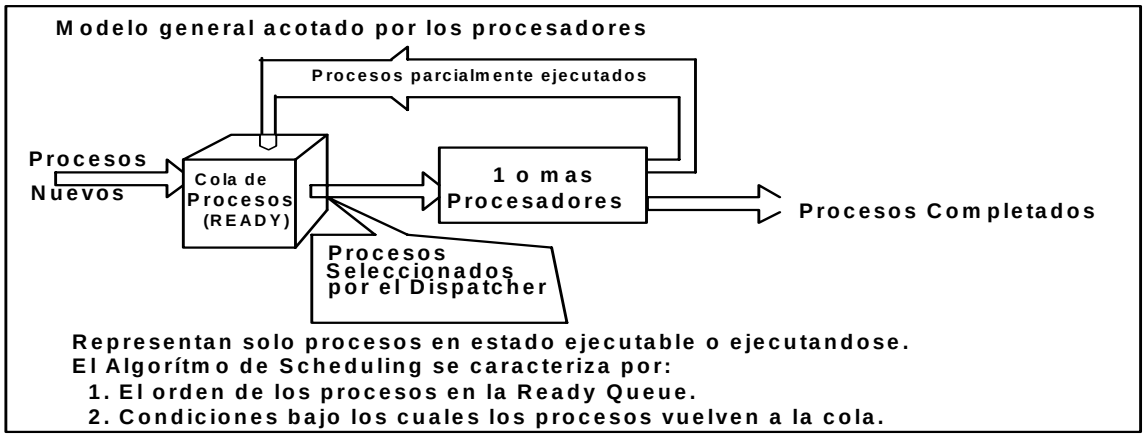


Fig. 3.11 Instancias en la planificación del corto plazo.

Resumiendo los planificadores que actúan sobre la cola de listos son long term scheduler, medium term scheduler (que ingresan procesos a la cola) y el short term scheduler (que saca procesos de la cola). La característica de la cola es que en multiprogramación nunca está vacía (al menos hay un proceso en ella: el Nulo).

El Objetivo general de un algoritmo de Planificación (Scheduling) es distribuir los trabajos y llevarlos a cabo por el Sistema para satisfacer las expectativas de los Usuarios y fundamentalmente, depende del Sistema y de la modalidad del procesamiento (batch, interactivo, etc.) mantener una carga de trabajo para que los recursos estén bien aprovechados. Esto determina el algoritmo de planificación.

La conmutación del procesador entre un proceso y otro se denomina **context switch** (cambio de contexto de ejecución según lo explicado).

- Context switch involucra:

1. Preservar el estado del proceso viejo (guardar en el stack su PCB).
2. Recuperar el estado del proceso nuevo (recuperar su PCB).
3. Bifurcar a la dirección donde había sido suspendido el nuevo proceso.

- Context switch es overhead puro. Debe ser lo mas rápido posible. Algunos valores típicos oscilan entre 1 y 100 μseg (algunos sistemas están en los nseg.) que se conoce como latencia en el despacho (***dispatch latency)***.

En la figura 3.11 indicamos los componentes del corto plazo: la cola de listos, el algoritmo seleccionador (traffic controller) y la rutina del Kernel que conmuta el contexto de ejecución (dispatcher) y presentamos un modelo general que resume el funcionamiento del procesamiento en el corto plazo

OBSERVACIÓN: Puede haber S. O. con otros esquemas o niveles de Schedulers. Existen S.O. cuyos diseños difieren de los tres niveles comentados y proveen otros o distintos niveles de Planificadores.

3.3. Criterios de Planificación de los Trabajos y de los Procesos

El **Planificador (scheduler)** es el módulo del S.O. que decide que proceso se debe ejecutar, para ello usa un algoritmo de planificación que debe cumplir con los siguientes objetivos:
Ser justa:

- ❖ Todos los procesos son tratados de igual manera.
- ❖ Ningún proceso es postergado indefinidamente.
- ❖ Maximizar la capacidad de ejecución: Maximizar el número de procesos servidos por unidad de tiempo.
- ❖ Maximizar el número de usuarios interactivos que reciban unos tiempos de respuesta aceptables: En un máximo de unos segundos.

- ❖ Ser predecible: Un trabajo dado debe ejecutarse aproximadamente en la misma cantidad de tiempo independientemente de la carga del sistema.
- ❖ Minimizar la sobrecarga: No suele considerarse un objetivo muy importante.
- ❖ Equilibrar el uso de recursos: Favorecer a los procesos que utilizarán recursos infrautilizados.
- ❖ Equilibrar respuesta y utilización: La mejor manera de garantizar buenos tiempos de respuesta es disponer de los recursos suficientes cuando se necesitan, pero la utilización total de recursos podrá ser pobre.
- ❖ Evitar la postergación indefinida: Se utiliza la estrategia del "envejecimiento".
- ❖ Mientras un proceso espera por un recurso su prioridad debe aumentar, así la prioridad llegará a ser tan alta que el proceso recibirá el recurso esperado.
- ❖ Asegurar la prioridad: Los mecanismos de planificación deben favorecer a los procesos con prioridades más altas.
- ❖ Dar preferencia a los procesos que mantienen recursos claves: Un proceso de baja prioridad podría mantener un recurso clave, que puede ser requerido por un proceso de más alta prioridad. Si el recurso es no apropiativo, el mecanismo de planificación debe otorgar al proceso un tratamiento mejor del que le correspondería normalmente, puesto que es necesario liberar rápidamente el recurso clave.
- ❖ Dar mejor tratamiento a los procesos que muestren un "comportamiento deseable": Un ejemplo de comportamiento deseable es una tasa baja de paginación.
- ❖ Degradarse suavemente con cargas pesadas:
- o Un mecanismo de planificación no debe colapsar con el peso de una exigente carga del sistema.
- o Se debe evitar una carga excesiva mediante las siguientes acciones:
  - No permitiendo que se creen nuevos procesos cuando la carga ya es pesada.
  - Dando servicio a la carga más pesada al proporcionar un nivel moderadamente reducido de servicio a todos los procesos.

Nótese que algunas de estas metas también son contradictorias entre sí, por lo tanto se busca un balance entre ellas, por ejemplo, para ofrecer más tiempo de procesador a un usuario, hay que quitárselo a otro.

Se tiene dos instancias para organizar la ejecución de los procesos: la llegada de trabajos al job queue y la selección de procesos para su ejecución. Por lo que actúan dos planificadores básicamente con criterios diferentes:

- El long term scheduler que contempla el uso del procesador en los aspectos políticos: modalidad de uso Batch, interactivo, Tiempo Real, etc.
- El cortísimo plazo que contempla el aprovechamiento del procesador, o sea, su eficiencia.

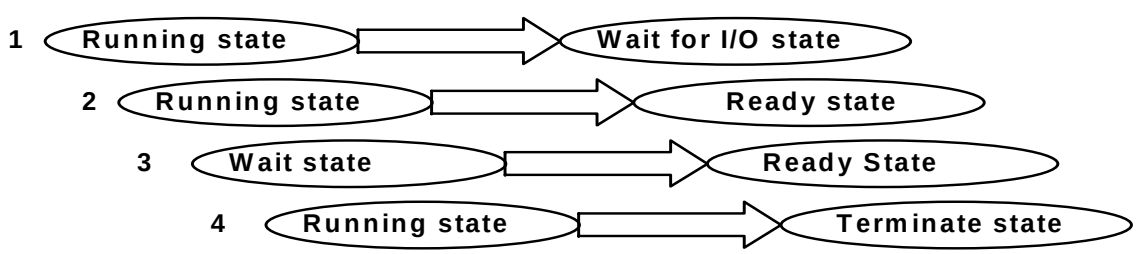


Fig. 3.12 Instancias en que se producen cambios de estado de un proceso.

Sobre la base de estos cambio de estados se utiliza dos esquemas de planificación:

1. **Nonpreemptive scheduling** (planificación sin sustitución, o sin reemplazo, o apropiativo, o no expropiativo en el uso del procesador):
   Los procesos usan al procesador hasta que terminan o hacen un I/O (Casos: 1 y 4 de la Fig. 3.12). También conocido como cooperative multitasking. Esta política de ejecución para terminación fue implementada en los primeros sistemas de lote (batch). Se permite que un proceso ejecute todo el tiempo que desee con uso exclusivo del procesador. Cuando ese proceso termina o se detiene por una E/S entonces se le da paso al siguiente.
2. **Preemptive scheduling** (planificación con sustitución o reemplazo o sin apropiación, o expropiativo en el uso del procesador):
   Los procesos usan al procesador hasta que quieren o hasta que el S.O. decide sustituirlos por otro proceso (casos: 1, 2, 3 y 4 de la Fig. 3.12). Generalmente conocida como política de **Planificación por torneo (Round Robin)** donde un proceso puede ser suspendido momentáneamente para ceder paso a otro esperando a ser ejecutado, logrando de esta forma multiplexar al procesador entre varios procesos en forma "casi simultánea".

El S.O. toma sus decisiones de planificación del procesador en uno de los siguientes momentos en que se producen cambios en los estados de un proceso.

3.3.1. Política vs. Mecanismo.

Hasta ahora hemos supuesto en forma tácita que todos los procesos en el sistema pertenecen a diversos usuarios y que, por tanto, compiten por el uso del procesador. Aunque esto es muy frecuente, a veces ocurre que un proceso tiene muchos hijos ejecutándose bajo su control. Por ejemplo, un proceso en un sistema de administración de una base de datos podría tener muchos hijos. Cada hijo podría trabajar una solicitud distinta, o bien, cada uno podría desarrollar cierta función (cálculos, acceso a disco, etc.).

Es completamente posible que el proceso principal tenga una idea excelente de cuáles de sus hijos son los más importantes (o críticos respecto al tiempo), y cuáles los menos. Por desgracia, ninguno de los planificadores analizados antes acepta datos de los procesos del usuario relativos a decisiones de planificación. Como resultado, el planificador pocas veces hace la mejor elección.

La solución a este problema es separar el **mecanismo de planificación** de la **política de planificación.** Lo que esto quiere decir es que el algoritmo de planificación queda parametrizado de alguna manera, pero los parámetros pueden ser determinados por medio de procesos del usuario.

Consideremos de nuevo el ejemplo de la base de datos. Supongamos que el kernel utiliza un algoritmo de planificación, pero que proporciona una llamada al sistema por medio de la cual un proceso puede establecer (y modificar) la prioridad de sus hijos. De esta forma, el padre puede controlar en detalle la forma de planificar sus hijos, incluso aunque él mismo no realice la planificación. En este caso, el mecanismo está en el kernel pero la política queda establecida por el proceso del usuario.

3.3.2. La planificación de los Trabajos.

En la Fig. 3.05 se planteaba la pregunta ¿dónde colocar los trabajos cuando se reciben?. Esta tarea la realiza el Long term Scheduler o Job Scheduler. Para ello utiliza básicamente los criterios de uso del procesador, si es non preemptive como en el caso de Batch o preemptive en sistemas interactivos.

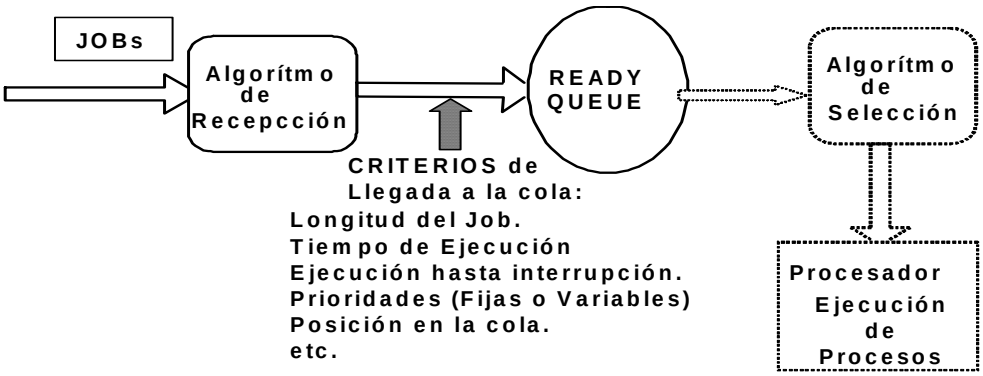


Fig. 3.13 Criterios de ingreso de trabajos para un solo procesador

Los Algoritmos para planificar la llegada a la cola de Listos se basan en políticas que son los siguientes:

Si bien hablamos de Job's, generalmente la entrada de los trabajos se realiza en modo Batch por lo que es común llamarlos procesos. Lo diferenciamos por la llegada a la cola de Listos para ejecutar.

a) FCFS (First-Come First-Served) - Primero en Llegar, Primero en ser Servido

- Es el algoritmo de planificación más sencillo.
- La implementación del FCFS se realiza fácilmente mediante una cola FIFO. Cuando un trabajo entra en la cola de Jobs o de preparados para crear procesos nuevos.
- Los procesos nuevos o listos para la ejecución (Ready Queue), su bloque de control de proceso (PCB) se enlaza al final de la cola.
- El código para la planificación FCFS es sencillo de escribir y de comprender.
- Sin embargo, las prestaciones del FCFS son, con frecuencia, bastante pobres.

Los Problemas que presenta son:

1. El tiempo medio de espera suele ser elevado.
2. Bajo nivel de utilización del procesador
3. Pobre tiempo de respuesta en procesos cortos en esquemas con mucha carga.

Veamos el caso:

Tiempo de espera

Consideremos que los trabajo s P_1, P_2 y P_3 están LISTOS para ejecutar su siguiente fase de CPU, cuya duración será de 24, 3 y 3 milisegundos, respectivamente. Si ejecutan en el orden P_1, P_2, P_3, entonces los tiempos de espera son: 0 para P_1, 24 para P_2 y 27 para P_3, o sea, en promedio, 17 ms. Pero si ejecutan en orden P_2, P_3, P_1, entonces el promedio es sólo 3 ms. En consecuencia, FCFS no asegura para nada que los tiempos de espera sean los mínimos posibles; peor aún, con un poco de mala suerte pueden llegar a ser los máximos posibles.

Utilización de CPU

Ahora supongamos que tenemos un trabajo intensivo en CPU y varios trabajos intensivos en I/O. Entonces podría pasar lo siguiente: el trabajo intensivo en CPU toma la CPU por un período largo, suficiente como para que todas las operaciones de I/O pendientes se completen. En esa situación, todos los trabajos están LISTOS, y los dispositivos desocupados. En algún momento, el trabajo intensivo en CPU va a solicitar I/O y va a liberar la CPU. Entonces van a ejecutar los otros trabajos, pero como son intensivos en I/O, van a liberar la CPU muy rápidamente y se va a invertir la situación: todos los trabajo s van a estar BLOQUEADOS, y la CPU desocupada. Este fenómeno se conoce como **efecto convoy**, y se traduce en una baja utilización tanto de la CPU como de los dispositivos de I/O. Obviamente, el rendimiento mejora si se mantienen ocupados la CPU y los dispositivos (o sea, conviene que no haya colas vacías).

b) SJF - Shortest Job First (también llamado SJN - Shortest Job Next)

Este algoritmo sólo se utiliza a nivel del Planificador de Trabajos (Long-term scheduler) aunque existe una adaptación del mismo para el corto plazo. El usuario debe predecir el tiempo de ejecución del proceso. EL SJF es probadamente óptimo, en el sentido de que produce el mínimo tiempo medio de espera para un conjunto dado de trabajos. El SJF es un caso especial del algoritmo general de planificación por prioridad.

La experiencia muestra que dando preferencia a un trabajo corto sobre uno largo, se reduce el tiempo de espera del trabajo corto más de lo que se incrementa el tiempo de espera del largo. Por consiguiente, el tiempo de espera medio, decrece. La dificultad del SJF consiste en conocer la longitud a priori que puede ser implementada en tiempo de compilación o después de una corrida.

Para la planificación de trabajos (a largo plazo) en un sistema batch, podemos utilizar el límite de tiempo por trabajo. De esta manera se motiva a los usuarios estimar el límite de tiempo de sus trabajos con precisión, puesto que un valor inferior puede significar un retorno más rápido (un valor demasiado bajo ocasionará un error por "tiempo límite excedido" y requerirá el relanzamiento).

Este algoritmo puede producir inanición (***Starvation***). Un proceso que está listo para ejecutarse pero que no dispone del procesador o de algún recurso puede considerarse bloqueado, esperando al procesador. Un algoritmo de planificación por tamaño puede dejar algunos procesos mas largos esperando al procesador indefinidamente. Esto se conoce como inanición. En un sistema muy cargado, una corriente persistente de procesos cortos pueden impedir que un proceso largo llegue a conseguir al procesador. Generalmente ocurrirá una de éstas dos cosas, o bien el trabajo terminará siendo ejecutado finalmente, o bien el Sistema Operativo perderá o suspenderá todos los trabajos largos inacabados.

Una solución a este problema de bloqueo de los trabajos largos es el ***envejecimiento (aging)***. Es una técnica que consiste en incrementar gradualmente la prioridad de los trabajos que esperan en el sistema durante mucho tiempo y así poder lograr el recurso procesador.

Supongamos que tenemos tres trabajos cuyos tiempos estimados de CPU son de *a*, *b* y *c* milisegundos de duración. Si ejecutan en ese orden, el tiempo medio de espera es:

$$(0 + a + (a + b))/3 = (2a + b)/3$$

O sea, el primer trabajo que se ejecute es el que tiene mayor incidencia en el tiempo medio, y el último, tiene incidencia nula. En conclusión, el tiempo medio se minimiza si se ejecuta siempre el trabajo con la menor próxima fase de CPU que esté LISTO. Además, es una buena manera de prevenir el efecto convoy. Lo malo es que para que esto funcione, hay que adivinar el futuro, pues se requiere conocer la duración de la próxima fase de CPU de cada trabajo. Lo que se hace es predecir la próxima fase de CPU en base al comportamiento pasado del trabajo, usando un *promedio exponencial*. Supongamos que

nuestra predicción para la *n*-ésima fase es T_n, y que en definitiva resultó ser t_n. Entonces, actualizamos nuestro estimador para predecir T_{n+1}

$$T_{n+1} = (1-\alpha)\, t_n + \alpha\, T_n$$

El parámetro *alpha* (α), entre 0 y 1, controla el peso relativo de la última fase en relación a la historia pasada.

$$T_{n+1} = (1-\alpha)t_n + \alpha\,(1-\alpha)t_{n-1} + \ldots + \alpha^j(1-\alpha)t_{n-j} + \ldots + \alpha^{n+1}T_0$$

O sea, mientras más antigua la fase, menos incidencia tiene en el estimador.
Un valor atractivo para *alpha* es 1/2, ya que en ese caso sólo hay que sumar los valores y dividir por dos, operaciones especialmente fáciles en aritmética binaria. Esta cuestión se repite en el algoritmo SPF en el corto plazo.

c) Planificación por prioridad:

Cada trabajo tiene asociada una prioridad, y la primera posición se asigna al trabajo que tiene la prioridad más alta para ingresarlo en la cola de Listos. Los trabajos con la misma prioridad se planifican como FCFS. Por lo que la cola tendrá distintos niveles, tantos como niveles de prioridad se establecieron en el sistema.

Un algoritmo de planificación por prioridades puede dejar algunos procesos de baja prioridad esperando por procesador indefinidamente por lo que valen las mismas consideraciones de **inanición (starvation**) explicados en el punto anterior. Entonces vale decir que en un sistema muy cargado, una corriente intensa de procesos de alta prioridad puede impedir que un proceso de baja prioridad llegue a conseguir al procesador que también puede ser resuelto mediante la técnica de aging.

Criterio	Algoritmo	
	Non Preemptive	Preemptive
Longitud del Job o Tiempo de ejecución	FCFS, SJN	---
Ejecución hasta interrupción	Interrupción	---
Prioridades Fijas o externas	FCFS, SJN	---
Prioridades Variables o internas	---	Cambio de posición en la cola
Posición en la cola	Multicolas	---

Tabla 3.1 criterios y algoritmos para la planificación de trabajos

Obsérvese que discutimos la planificación desde el punto de vista de alta prioridad y baja prioridad. Las prioridades consisten generalmente en una gama determinada de números, tales como 0 a 7, o 0 a 4095. Sin embargo, no existe un acuerdo general acerca de si 0 es la prioridad más alta o la más baja. Algunos sistemas utilizan los números inferiores para representar las prioridades bajas; otros dedican los números inferiores a las prioridades altas como es el caso de UNIX. Esta diferencia puede ocasionar alguna confusión por lo que se deberá ver como se implementa en cada S.O..

Una óptica es ver a los procesos utilizando el recurso procesador y la planificación de estos procesos. Una segunda óptica es ver el procesador recibiendo los procesos. Describiremos ambas ópticas y sus criterios.

Hay muchos criterios para definir la prioridad. Por ejemplo:

1. Según categoría del usuario.
2. Según tipo de trabajo o proceso: sistema, interactivo, o por lotes; o bien, intensivo en CPU o intensivo en I/O.
3. Según cuanto hayan ocupado la CPU hasta el momento
4. Para evitar que un proceso de baja prioridad sea postergado en demasía, aumentar prioridad mientras más tiempo lleve esperando: envejecimiento (*aging*).
5. Para evitar que un proceso de alta prioridad ejecute por demasiado tiempo, se le puede ir bajando la prioridad.
6. etc.

SJF es planificación por prioridad donde la prioridad es función del estimador de la duración de la próxima fase de CPU por eso es un caso especial de planificación por prioridad.

3.3.3. La planificación de los Procesos.

El planificador en el corto plazo se ejecuta cuando ocurre un suceso que puede conducir a la interrupción del proceso actual o que ofrece la oportunidad de expulsar de la ejecución al proceso actual a favor de otro. Responde a políticas de ejecución, o sea, uso del Procesador [4].
Como ejemplos de estos sucesos se tienen:

- Interrupciones de reloj
- Interrupciones de E-S
- Llamadas al sistema operativo
- Señales

Los Criterios de planificación generalmente se refieren al uso del Procesador. Los algoritmos de planificación del procesador tienen diferentes propiedades y podrían favorecer a una clase de procesos más que a otra. Por lo tanto se deben tener en cuenta ciertos criterios para que la comparación de algoritmos tienda hacia la elección de la solución óptima:

a) Orientados al Usuario

Se refieren al comportamiento del sistema tal y como lo perciben los usuarios y los procesos individuales.

- ***Tiempo de Respuesta:*** es el intervalo de tiempo transcurrido desde el momento que se emite una solicitud hasta que comienza a recibir la respuesta.
- ***Tiempo de Retorno:*** Es el intervalo de tiempo transcurrido entre el lanzamiento de un proceso y su finalización.
- ***Plazos:*** Cuando se pueden especificar plazos de terminación de un proceso, la disciplina de planificación debe subordinar otras metas a la maximización del porcentaje de plazos cumplidos. Un determinado trabajo debe ejecutarse aproximadamente en el mismo tiempo y con el mismo costo sin importar la carga del sistema

Los Algoritmos de acuerdo a los criterios son los siguientes:

Criterio	Algoritmo	
	Non Preemptive	Preemptive
Longitud de los Procesos o Tiempo de ejecución	FCFS, SPN	SRTF(SRF)
Ejecución hasta interrupción	Int.	RR
Prioridades Fijas o externas	SPN, FCFS	---
Prioridades Variables o internas	---	Cambio de posición en la cola
Posición en la cola	Multicolas	---

Tabla 3.2 criterios y algoritmos para la planificación de Procesos

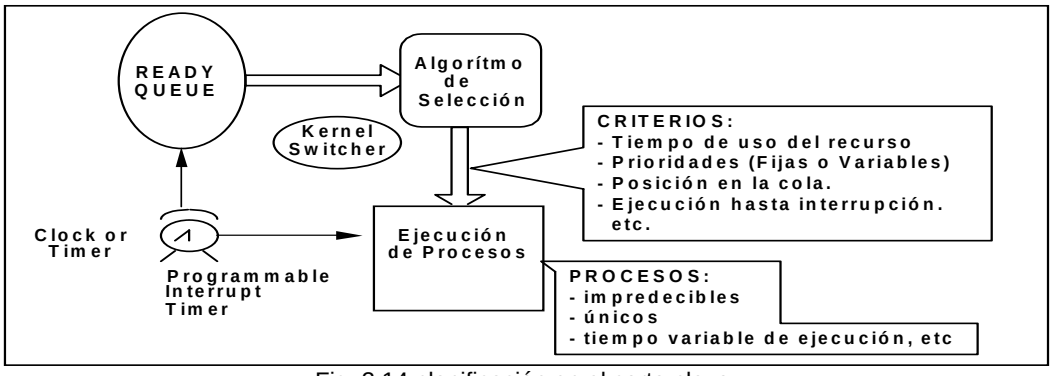


Fig. 3.14 planificación en el corto plazo.

b) Orientados al Sistema

Se centran en el uso efectivo y eficiente del procesador y abarca los siguientes criterios:

- ***Productividad:*** La política de planificación debe intentar maximizar el número de procesos terminados por unidad de tiempo.
- ***Utilización del procesador:*** Es el porcentaje de tiempo en el que el procesador está ocupado.

[4] Utilizaremos los vocablos Procesador y CPU como sinónimos. Cuando hablamos de planificación de procesador estamos refiriéndonos al monoprocesamiento, o sea un solo procesador.

- ***Equidad:*** Los procesos deben ser tratados de igual forma y ningún proceso debe sufrir inanición.
- ***Prioridades:*** La política de planificación debe favorecer a los de mayor prioridad.
- ***Equilibrio de Recursos:*** Se debe favorecer a los procesos que no utilicen recursos sobrecargados.

Comparación del rendimiento

El rendimiento de las distintas políticas de planificación es un factor crítico en la elección de una política. Sin embargo, es imposible hacer una comparación definitiva porque el rendimiento relativo depende una gran variedad de factores, incluyendo la distribución de probabilidad de los tiempos de servicios de los procesos, la eficiencia de la planificación y de los mecanismos de cambios de contexto, la naturaleza de las peticiones de E-S y el rendimiento del subsistema de E/S.

3.3.4. Algoritmos de planificación del Procesador

La planificación del procesador consiste en decidir a cuál de los procesos situados en la cola de preparados para la ejecución se le debe asignar la CPU. El procesador es un recurso muy escaso por lo que se trata que su uso se aproveche al máximo.

Hay muchos algoritmos de CPU diferentes. En épocas pasadas de los sistemas de procesamiento por lotes y cintas magnéticas, el algoritmo de planificación era sencillo: sólo había que ejecutar el siguiente trabajo en cinta. En los sistemas de multi-usuario de tiempo compartido, el algoritmo es más complejo.

Decíamos que los diferentes algoritmos tienen propiedades distintas que pueden favorecer a una clase de procesos sobre las demás. Al seleccionar el algoritmo a utilizar en una situación determinada, hay que tener en cuenta las propiedades y objetivos de los mismos.

Una vez seleccionado el criterio de comparación, que se de detallan más adelante en este punto, generalmente se busca optimizarlo. Es deseable maximizar la utilización de la CPU y la productividad o minimizar el tiempo de retorno, el tiempo de espera y el tiempo de respuesta. En la mayoría de los casos lo que se optimiza es la media. No obstante, a veces puede ser deseable optimizar los valores mínimos o máximos, más que la media. Por ejemplo, para garantizar que todos los usuarios obtengan un buen servicio, se podría minimizar el tiempo de respuesta máximo. En otro caso se buscará balacear la carga de trabajo del sistema.

También se ha sugerido que en los sistemas interactivos (como en los sistemas en tiempo compartido), es más importante minimizar la *varianza* del tiempo de respuesta que minimizar el tiempo medio de respuesta. Un sistema con un tiempo de respuesta razonable y *predecible* puede considerarse mejor que un sistema más rápido en promedio pero altamente variable. Se ha trabajado muy poco en los algoritmos de planificación de CPU para minimizar la varianza.

Generalmente los algoritmos se basan en la siguiente propiedad de los procesos:

Durante su ejecución, un proceso alterna entre períodos de uso de CPU (***CPU bursts***) o ráfagas de CPU y períodos de uso de E/S (***I/O bursts***) Los CPU burst pueden estar limitados por tiempo o por cantidad de instrucciones esto se conoce como ***CPU bound***.

- La ejecución siempre comienza y termina por un CPU burst.
- En sistemas con procesos E/S bound (limitado por E/S), la duración de los CPU bursts suele ser muy corta.
- En sistemas con procesos CPU bound, la duración de los I/O bursts es acotada de acuerdo al quantum de tiempo otorgado al proceso.

Los algoritmos deben ser seleccionados de acuerdo al tipo de procesos que se desean ejecutar. Se han sugerido muchos criterios para la **comparación de algoritmos** de planificación de CPU. Las características que se utilicen en la comparación pueden introducir diferencias substanciales en la determinación del mejor algoritmo.

Entre los criterios mas comunes que se emplean están los siguientes:

1. Equidad: garantizar que cada proceso obtiene su proporción justa de la CPU.
2. Eficacia: mantener ocupada la CPU el 100% del tiempo en lo posible. La utilización de la CPU va desde 0 a 100% teórico y en la realidad se utiliza entre el 40% -baja carga- al 90% -alta carga-saturación.
3. Tiempo de respuesta: minimizar el tiempo de respuesta para los usuarios interactivos (***response time***). Es el lapso de tiempo desde que el usuario hizo su entrada al sistema hasta que obtuvo su primer resultado.

4. Tiempo de retorno: minimizar el tiempo que deben esperar los usuarios por lotes para obtener sus resultados.
5. Rendimiento: maximizar el número de tareas procesadas por hora, o sea, la cantidad de procesos ejecutados en una unidad de tiempo (***throughput***)
6. Tiempo de ejecución desde que se somete el nuevo trabajo hasta que el proceso muere (***turnaround time o tiempo de servicio***).
7. Tiempo de espera en la Ready Queue (***waiting time***).

- **Medidas:**

Consideremos los siguientes tiempos que incurre un proceso:

t_E : ***Tiempo*** neto ***de ejecución*** del proceso (uso neto de CPU).

t_f : ***Tiempo*** cronológico en que ***finaliza*** su ejecución en el sistema de cómputo.

t_i : ***Tiempo*** cronológico en que ***ingresa*** al sistema.

*El **tiempo de servicio** será:* $t_s = t_f - t_i$	(1)
El tiempo de ***espera*** será $E = t_s - t_E$	(2)

y definimos como ***índice de servicio*** la relación del tiempo de ejecución y el tiempo de vida (t_S) del proceso en el sistema. Si ésta relación tiende a 1 se dice que el proceso utiliza mucho CPU burst y si tiende a cero usa I/O burst.

$$I_s = \frac{t_E}{t_s} \quad (3)$$

(1), (2) y (3) son válidos para un solo proceso. Para más de un proceso se utiliza los promedios de cada uno.

tiempo medio de servicio : $s = \frac{1}{n} \sum_{i=1}^{n} (t_f - t_i)_i$ (4)

Análogamente se calcula el tiempo medio de espera y la Eficiencia (índice medio de servicio).

3.4. Algoritmos NON-PREEMPTIVE (sin reemplazo o apropiativos)

Recordemos que se refiere a que los procesos toman posición del procesador y no lo abandonan salvo que se completen o hagan una operación de Entrada/Salida. Es decir que no se interrumpe su uso de la CPU cuando llega un proceso de mayor prioridad y tampoco se atienden las interrupciones por reloj.

El FCFS, el SPF y los algoritmos por prioridad son algoritmos de planificación *apropiativos (non-preempive)* en el uso del procesador ya que se apoderan de ella y no la liberan (se **apropian** de ella) por eso se dice que no se los sustituyen en el uso. Una vez asignada la CPU a un proceso, este puede mantenerla hasta que desee liberarla, bien por haber terminado o bien por solicitar una E/S (si el algoritmo contempla el reemplazo de los procesos caso contrario el procesador queda asignado a ese proceso y espera ocioso).

3.4.1. FCFS (First-Come First-Served)

Con mucha diferencia, el algoritmo de planificación de CPU más sencillo es el First-Come-First-Served (FCFS). Esto es, el primer proceso en solicitar la CPU es el primero en recibir la asignación de la misma. La implementación del FCFS se realiza fácilmente mediante una cola FIFO (primero entrado, primero

salido). Cuando un proceso entra en la cola de preparados o listos para la ejecución (o sea que está en la Ready Queue), su bloque de control de proceso (PCB) se enlaza al final de la cola.

Cuando el procesador queda libre, éste se asigna al proceso situado al principio de la cola. Entonces el proceso en ejecución se elimina de la cola. El código para la planificación FCFS es sencillo de escribir y de comprender. Sin embargo, las prestaciones del FCFS son, con frecuencia, bastante pobres.

El primer proceso en requerir la CPU, la obtiene y la utiliza. Es un algoritmo muy fácil de implementar usando una cola FIFO.
Los Problemas que presenta son:
1. El tiempo medio de espera suele ser elevado.
2. Bajo nivel de utilización de la CPU
3. Pobre tiempo de respuesta en procesos cortos en esquemas con mucha carga..

El FCFS es intrínsecamente un algoritmo apropiativo en el uso del Procesador.

3.4.2. SPF-Shortest Process First (también llamado SPN-Shortest Process Next)

Otro enfoque a la planificación de CPU le corresponde al algoritmo Shortest-Process-First (SPF) o Next (SPN), el cual asocia a cada trabajo la longitud de su siguiente ráfaga de procesador (CPU burst). Cuando el procesador queda libre, se asigna al proceso que tenga la siguiente ráfaga de CPU más pequeña. Si dos procesos tienen la misma ráfaga de CPU, se utiliza el FCFS. Obviamente que se requiere obtener la información de la longitud de la ráfaga para realizar la planificación y este valor es relativamente difícil determinar o, en el peor de los casos, no hay manera de saber la longitud de la siguiente ráfaga de CPU. Un enfoque que resuelve este problema consiste en tratar de aproximar la planificación del SPF.

- Para adaptarlo a un short-term scheduler, se trata de predecir el siguiente CPU burst, se lo predice generalmente tomando la media exponencial de los CPU burst previos. Para ello utilizamos la fórmula (5).
- SPF también puede ser preemptive. Es decir un proceso puede ser interrumpido en su CPU burst (imposible de hacerlo en el esquema non-preemptive).
- En ese caso, se elige el proceso al que le queda menos tiempo para finalizar su CPU burst (**s**hortest-**r**emaining-time first).

Se puede predecir el valor la longitud de la siguiente ráfaga de CPU suponiendo que la ráfaga siguiente de CPU será similar en longitud a las anteriores. Así, calculando una aproximación de la longitud de la ráfaga de CPU siguiente, se puede seleccionar el proceso que tenga la ráfaga predicha más corta. La ráfaga de CPU siguiente se predice generalmente como una media exponencial de las longitudes medidas de la ráfagas de CPUs previas. Sea T_n la longitud de la n-ésima ráfaga de CPU y T_{n+1} nuestro valor calculado de la ráfaga de CPU siguiente. Entonces, para α, siendo $0 \geq \alpha \geq 1$, se define:

$$T_{n+1} = \alpha \cdot l_n + (1 - \alpha)\, T_n \qquad (5)$$

Donde:

T_n : Longitud estimada del tiempo de uso de CPU anterior.
l_n : Longitud real del CPU burst anterior.
T_{n+1} : Longitud estimada del siguiente CPU burst.
α : Factor de ponderación (usualmente 50%).

Esta fórmula define una media exponencial. El valor de T_n contiene nuestra información más reciente o actual; T_n almacena la historia pasada en el instante actual. El parámetro α controla el peso relativo de la historia reciente y de la anterior en nuestra predicción.

Si $\alpha = 0$, entonces $T_{n+1} = T_n$, y la historia reciente no tiene efecto (las condiciones actuales se supone que son transitorias); si $\alpha = 1$, entonces $T_{n+1} = l_n$ y por consiguiente lo único que importa son las ráfagas de CPU más recientes (se supone que la historia es irrelevante). Más corrientemente, $\alpha = 1/2$, de modo que la historia reciente y la más antigua pesan igual.

Se asigna la CPU al proceso que use la misma por menos tiempo, esto implica que el tiempo medio de espera es mínimo.

Expandir la fórmula para t_{n+1} posibilitará comprender el comportamiento del promedio exponencial:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1} \tau_0 \quad (6)$$

Dado que tanto α como (1-α) son menores o iguales a 1, cada término sucesivo tiene menos peso que su predecesor.

Un riesgo que se corre utilizando este algoritmo es la posibilidad de que los procesos grandes no lleguen a ejecutarse (stravation) si ingresan continuamente procesos cortos.

Como observación importante es de resaltar la pérdida de tiempo que se emplea para efectuar este cálculo por lo que no se utiliza este algoritmo.

El algoritmo SPF (Shortest Process First) puede ser un algoritmo por prioridades en el que la prioridad (*p*) es la inversa de la siguiente ráfaga de CPU predicha (*T*), p=1/*T*. Cuanto más larga sea la ráfaga de CPU, menor la prioridad y viceversa

3.4.3. Planificación por prioridad:

Las prioridades pueden definirse bien internamente o bien externamente. Las prioridades definidas internamente utilizan alguna cantidad o cantidades mensurables para calcular la prioridad de un proceso. Por ejemplo, en el cálculo de prioridades se han utilizado los límites de tiempo, el requerimiento de memoria, el número de archivos abiertos y la razón entre la media de las ráfagas de E/S y la media de las ráfagas de CPU, tiempos de uso de Procesador en un polinomio, etc.. Las prioridades externas se establecen mediante criterios ajenos al sistema operativo, tales como lo que se paga por el empleo del computador, el departamento que patrocina el trabajo, y otros factores externos, con frecuencia políticos.

Las prioridades pueden ser:

Internas o **Dinámicas**: modificables por el S.O. en ejecución mediante uno o más parámetros medibles.

Externas o **Estáticas**: puestas arbitrariamente por el centro de cómputos de acuerdo a factores externos al sistema. Esto puede representar peligro de **bloqueo por inanición** o **starvation** Para evitar este bloqueo se usa la ***técnica de envejecimiento*** o **aging**: Incrementar gradualmente la prioridad de los procesos que han estado esperando mucho tiempo como lo hace el S.O. UNIX.

Un proceso puede entrar con una prioridad fija y luego ser modificada, ya sea por el propio usuario o por el S.O.

Resumiendo : Se asocia un número a cada proceso, ese número indica la prioridad con que será seleccionado para su ejecución. El significado de los números es arbitrario. Algunos S.O. consideran a mayor número mayor prioridad, en cambio otros a menor número, mayor prioridad.

Si asumimos que a mayor número, mayor prioridad, por ejemplo, en el caso de SPF, utilizaremos la siguiente fórmula:

$$p = \frac{1}{T_{n+1}} \quad (7)$$

a mayor $T_{n+1} \Rightarrow$ menor prioridad

3.5.4. HRRN (High Response Ratio Next)

Los algoritmos SPF/SRT tienden a producir bloqueos por inanición en los procesos largos, por lo que se introducen técnicas de envejecimiento. Un método que se aplica para realizar esto es el siguiente:

$$\textbf{Prioridad} = \frac{\textbf{Tiempo de espera + Tiempo de ejecución}}{\textbf{Tiempo de ejecución}} \quad (8)$$

Esta fórmula es interesante ya que permite conocer la “edad” del proceso, así aunque se siga utilizando el criterio del proceso más corto primero, se puede tener la certeza de que los procesos largos se ejecutaran.

La decisión de planificación se basa en una estimación del tiempo de retorno normalizado. Cuando el proceso actual termina o se bloquea, se elige el proceso listo con un valor mayor de HRRN. Favorece a los procesos cortos.

El envejecimiento sin que haya inanición incrementa el valor de la razón, de forma que los procesos más largos pasen finalmente primero. El tiempo esperado de servicio debe estimarse antes de emplear la técnica de la mayor tasa de respuesta.

En este caso se intenta que las proporciones de tiempo τ sean reducidas al máximo, este algoritmo comparado con el anterior, no necesita tener con exactitud el tiempo de ejecución total, solo una aproximación, y luego lo va corrigiendo durante la corrida del proceso.

Se basa en un algoritmo que selecciona de la cola de procesos listos a aquel proceso que tiene la relación de respuesta mayor, tomando como dicho valor el siguiente:

$$\tau = \frac{w + s}{s} \qquad (9)$$

τ = relación de respuesta
w = tiempo consumido esperando por el procesador
s = tiempo de servicio esperado

3.5. Algoritmos PREEMPTIVE (con reemplazo en el uso de la CPU)

Estos algoritmos liberan al procesador del proceso que lo esta usando para seleccionar uno se la cola de listos como reemplazo. Dado que el proceso por sí mismo no deja el uso del procesador, se debe incorporar un mecanismo para forzar un context switch. Generalmente se recurre a una interrupción, por ejemplo, de reloj que requiere una atención del Kernel y este aprovecha esta circunstancia para cambiar el PCB del proceso en uso del procesador por otro PCB de la cola de Listos.

3.5.1. Round Robin (RR) o torneo cíclico.

Un proceso puede salir del estado de ejecución por tres motivos:

a) que termine su ejecución,
b) se proceda al llamado a una entrada – salida y el proceso se quede bloqueado y
c) que se supere el quantum de ejecución del proceso, se dispare la interrupción del reloj y sea automáticamente retirado del estado de ejecución.

Luego el dispatcher deberá seleccionar de la cola de ejecución otro proceso, que lo hará de acuerdo a una secuencia de anillo que irá girando siempre en un sentido, ejecutando un tiempo para cada proceso de la misma.

En este caso, como es un algoritmo no apropiativo, se realizan sobre los procesos que están ejecutándose sobre el procesador interrupciones, que interrumpen la ejecución del proceso actual, colocando un nuevo proceso en su lugar, durante un tiempo predefinido.

El algoritmo de planificación *round-robin (RR)* fue especialmente diseñado para sistemas de tiempo compartido. Se define una pequeña unidad de tiempo común llamada ***quantum de tiempo o time slice (QT)*** (porción de tiempo), que generalmente tiene un valor entre 10 y 100 milisegundos (en el caso de UNIX es de 1 segundo). La cola de listos se trata como una cola circular. El planificador de CPU recorre la cola, asignando procesador a cada proceso durante un intervalo de tiempo de hasta un quantum.

Para implementar la planificación RR, la cola se mantiene como una cola de procesos FIFO. Los procesos nuevos se añaden al final de la cola. El planificador de la CPU selecciona el primer proceso de la cola, fija un temporizador para interrumpir, tras cumplirse un quantum de tiempo, se produce la interrupción por reloj que despacha el proceso a la cola de listos y selecciona otro proceso.

No siempre ocurre una interrupción para ser removido un proceso de la CPU. Hay casos en que el proceso puede tener una ráfaga de CPU (**CPU burst**) inferior a un quantum de tiempo debido a que se completó o requiere de una operación de Entrada/Salida, en este caso, el proceso mismo libera la CPU

voluntariamente, emitiendo un pedido de E/S o una llamada al sistema de terminando. Entonces se pasa al proceso siguiente en la cola.

En muchos casos, la ráfaga de CPU del proceso en ejecución es mayor que el quantum de tiempo, el temporizador sobrepasará su límite y ocasionará una interrupción al sistema operativo. Los registros del proceso interrumpido se guardan en su bloque de control de proceso, y el proceso se pone al final de la cola. Entonces el planificador selecciona el proceso siguiente de la cola y le asigna el quantum de tiempo siguiente.

En el algoritmo de planificación Round Robin puro no se asigna la CPU a ningún proceso durante más de un quantum de tiempo consecutivo si hay otros procesos en la cola de Listos. Si la cola esta vacía seguirá ejecutando el proceso hasta que haya un nuevo proceso en la Ready Queue. Entonces, si la ráfaga de CPU excede un quantum de tiempo, pierde la CPU y se devuelve a la cola de listos por lo que RR es un algoritmo de planificación no apropiativo.

Si hay *n* procesos en la cola y el quantum de tiempo es *q*, entonces cada proceso obtiene 1/*n* del tiempo de CPU en fragmentos de al menos *q* unidades de tiempo cada vez. Cada proceso tiene que esperar no más de **(*n* - 1) * *q*** unidades de tiempo hasta su quantum de tiempo siguiente.

Las prestaciones de la planificación Round-Robin dependen fuertemente del quantum de tiempo. En un extremo, si el quantum de tiempo es muy largo (infinito), RR pasa a ser igual que FCFS. Si el quantum de tiempo es muy corto (digamos un microsegundo), RR recibe el nombre de *procesador compartido* y aparece (en teoría) como si cada uno de los *n* procesos tuviera su propio procesador ejecutando a una velocidad equivalente a **1/*n*** de la velocidad del procesador real.

Este enfoque fue utilizado en el hardware del CDC 6600 para simular 10 procesadores periféricos con un hardware de un único procesador y diez conjuntos de registros [Thornton 1970]. El hardware ejecuta una instrucción para un conjunto de registros, luego va al siguiente. Este ciclo prosigue, resultando diez procesadores lentos en lugar de uno rápido.

El conflicto surge en el momento de decidir la duración del Quantum de Tiempo (QT)para cada proceso:

a) Si el QT es muy pequeño, produce mucho overhead por la gran cantidad de cambios de contexto de ejecución que hace el Sistema Operativo.

b) Si el QT es muy grande produce un tiempo de reacción muy pobre porque los procesos en cola de listos esperan demasiado y si es infinito se convierte en FCFS.

En software, sin embargo, se tiene otros problemas. Concretamente, al final de cada quantum de tiempo, nos encontramos con una interrupción procedente del temporizador. El procesamiento de la interrupción para conmutar la CPU a otro proceso requiere salvar todos los registros del proceso antiguo y restaurar los registros del proceso nuevo. Dijimos que a este procedimiento lo llamamos ***cambio de contexto***. El cambio de contexto es un tiempo de trabajo extra del S.O. en que utiliza procesador, o sea que es puro **overhead**. Varía de una máquina a otra, en función de la velocidad de la memoria, del número de registros y de la existencia de instrucciones especiales.

Resumiendo: Inicialmente el RR fue diseñado para sistemas de tiempo compartido. La cola de listos, en este caso, es circular y a cada proceso se le da una pequeña unidad de tiempo para ejecutar, **time slice** o **time quantum** ($10 \leq TS \leq 100$ ms) que una vez fijado permanece constante, pues de él depende el rendimiento del Sistema.

- Se programa el timer para que interrumpa al final de cada time slice.
- Para que sea eficiente, la duración del context switch debe ser mucho menor que el time slice. Si es del orden del 10% del time slice en context switch se dice que el 10% del tiempo, la CPU, va a estar en context switch o sea ejecutando el S.O..
- Las prioridades de ejecución de los procesos se pueden implementar modificando el Time Slice de cada uno mediante un adecuado ajuste que puede ser realizada por el Súper- Usuario como en el caso de UNIX.

El Round Robin admite algunas modificaciones al mecanismo básico que implementan algunos S.O.. Estas modificaciones son:

- Si un proceso fue interrumpido antes de que finalice su quantum, ese tiempo de procesamiento perdido lo recupera en su siguiente turno, o sea , que se le suma a su Quantum el remanente no utilizado en el turno anterior.
- A los procesos nuevos se les da un Time Slice inicial mayor que el normal.
- Cuando entra un proceso nuevo en la cola, en lugar de ponerlo al final, se lo inserta al principio de la misma.
- Si a un proceso le falta poco (digamos 5 quantums) se le otorga todos los Time Slice que necesite para su finalización.

Este algoritmo es particularmente efectivo en sistemas de tiempo compartido de propósitos generales, aunque presenta dificultades si existe una mezcla de procesos limitados por CPU y procesos

limitados por E/S. Esto se debe a que los procesos limitados por E/S tienen ráfagas de CPU muy cortas, por lo que usan el procesador por poco tiempo y quedan bloqueados, luego se ejecuta un proceso limitado por CPU, usando todo su cuantum y volviendo rápidamente a la cola de listos, mientras que el proceso de E/S pierde su turno, lo que produce una baja en la performance de los procesos de E/S, un uso ineficiente de los dispositivos y un aumento de la varianza del tiempo de respuesta. Para solucionar este problema se implementa un algoritmo llamado ***VRR (Virtual Round Robin)***, el cual funciona de manera similar al RR, aunque los procesos que son desbloqueados se colocan en una cola FIFO auxiliar, la cual tiene prioridad por sobre la cola principal, al momento de decidir que proceso pasa a ejecución se consulta esta cola auxiliar y si hay procesos en ella se procede a seleccionarlos primero.

3.5.2. Menor tiempo restante (SRT Shortest Remaining Time First)

Se puede definir fácilmente como la versión expropiativa del algoritmo SPF.

El planificador debe disponer de una estimación del tiempo de proceso para poder llevar a cabo la función de selección, existiendo el riesgo de inanición para procesos largos. Los Procesos cortos reciben una atención inmediata.

En este caso el planificador siempre selecciona primero para ejecutar al proceso con menor tiempo restante o remanente, o sea, selecciona al proceso al que le queda menos tiempo esperado de ejecución en el procesador, Un proceso puede ser expulsado cuando otro proceso esta listo. Cuando un nuevo proceso que ya ejecutó al menos una vez, se agrega a la cola de listos, puede tener un tiempo restante de ejecución, menor que el que se encuentra ejecutando actualmente, por lo que pasa a ejecución.

Al igual que con SPF se corre el riesgo de starvation de los procesos largos lo cual haría que el sistema no sea eficiente.. Otra ventaja que presenta es la de poseer la mejor performance, ya que los procesos más cortos toman inmediatamente el control de la CPU.

Cada vez que un proceso es agregado a la cola por el planificador de medio o de largo plazo, y tenga un tiempo restante de ejecución menor del que se encuentra ejecutando, es inmediatamente puesto en ejecución por el dispatcher y retirado el que se encontraba en ese estado.

A comparación con el Round Robin este algoritmo es más eficiente debido a que no se produce overhead muy frecuente como ocurre en él, debido a que las interrupciones no son producidos por el reloj del sistema.

3.5.3. Planificación con colas de múltiples niveles y Realimentación

En el caso en que no se pueda determinar el tiempo de ejecución, no es pueden utilizar los algoritmos anteriores que necesitan este parámetro, para estos casos se utiliza este algoritmo.

Este algoritmo se basa en que la planificación es del tipo expropiativo por quantum de tiempo y un mecanismo de prioridades dinámico. Cuando llega un proceso nuevo, este es colocado en la cola de mayor prioridad, luego de su primer ejecución, este es colocado en la cola de prioridad siguiente menor y así sucesivamente hasta que llega hasta la última cola.

Dentro de cada cola se utiliza el algoritmo de planificación FCFS, en el caso de la última se utiliza el algoritmo Round Robin.

En este algoritmo puede llegar a ocurrir Starvation en el caso de que entren frecuentemente procesos nuevos, debido a que al tener mayor prioridad, no llega a ejecutarse los procesos de las últimas colas. Para evitar esto, se utiliza un mecanismo de variación del quantum de tiempo de ejecución de los procesos de acuerdo a la cola en la que se encuentra. A la primera se le asigna un quantum, a la cola siguiente dos quantum y así hasta llegar al final que es la que mayor tiempo de ejecución tiene. O sea en general a la cola RQ_{ii} se le asigna 2^{ii} quantum de tiempo, de esta forma se trata de que menos procesos lleguen hasta la última cola sin terminar su ejecución.

En el caso de que ocurra Starvation, en un proceso que se quede sin ejecución en la última cola, se lo puede enviar nuevamente hasta la cola de mayor prioridad para que continúe su ejecución.

Establece un conjunto de colas de planificación y sitúa los procesos en las colas, teniendo en cuenta, entre otros criterios, el historial de ejecución. Favorece a los procesos más nuevos y cortos que a los mas viejos y largos. Un problema que presenta el sencillo esquema anterior es que el tiempo de retorno de los procesos mayores puede alargarse de forma alarmante.

Puede ocurrir inanición si llegan regularmente nuevos trabajos al sistema.

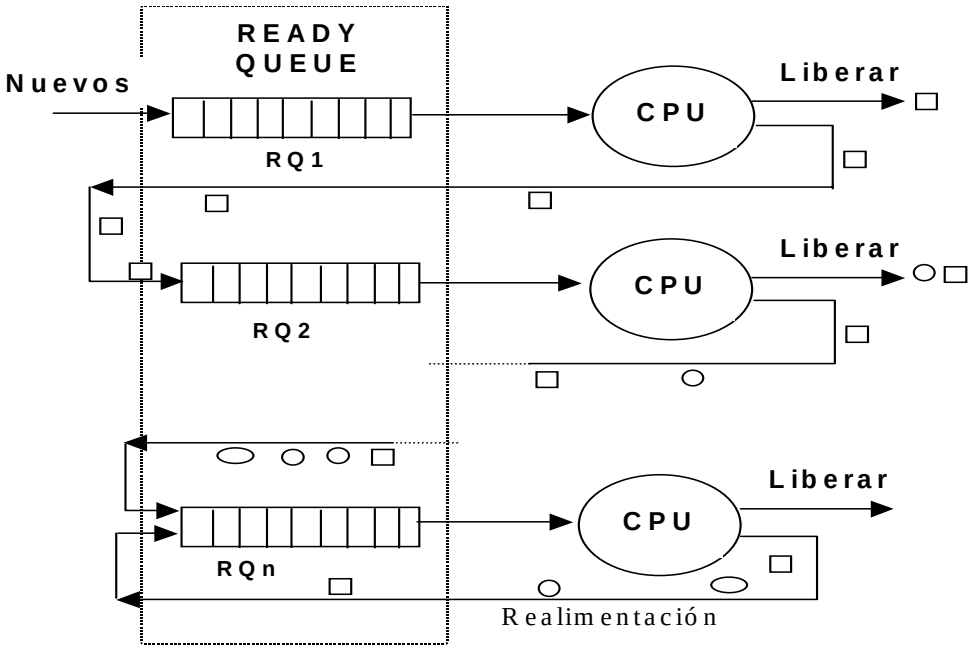


Figura 3.15. Colas de múltiples niveles con realimentación

Este tipo de planificación permite a un proceso pasar de una cola a otra. La idea es separar los procesos con diferentes características en cuanto a sus ráfagas de CPU. Si un proceso gasta demasiado tiempo en CPU, se le pasará a una cola con menor prioridad. Este esquema deja los procesos limitados por E/S y los procesos interactivos en las colas de más alta prioridad. Así mismo, si un proceso espera demasiado tiempo en una cola de baja prioridad pudiese pasarse a una de mayor prioridad, evitando así la inanición.

En general, un planificador de colas multinivel con realimentación está definido por los siguientes parámetros:

- El número de colas.
- El algoritmo de planificación para cada cola.
- El método empleado para determinar cuándo se debe promover un proceso a una cola de mayor prioridad.
- El método empleado para determinar cuándo se debe degradar un proceso a una cola de menor prioridad.
- El método empleado para determinar en cuál cola ingresará un proceso cuándo necesite servicio.

La definición de un planificador de colas multinivel con realimentación, lo convierte en el algoritmo de planificación de la CPU más general, ya que se puede configurar para adaptarlo a cualquier sistema especifico que se este diseñando. Desdichadamente, se requiere alguna forma de seleccionar valores para todos los parámetros de manera que se obtenga el mejor planificador posible.

Aunque este esquema es el más general, también es el más complejo.

3.5.4. Planificación con múltiples colas fijas

Uno de los primeros planificadores que utilizó prioridad era el S.O. CTSS de tiempo compartido (Corbato *et al*. , 1962). CTSS tenía el problema de que la alternancia entre procesos era muy lenta, puesto que la máquina 7094 (IBM) sólo podía mantener un proceso dentro de la memoria. Cada alternancia representaba un intercambio: el envío del proceso activo al disco y la lectura en el disco de un nuevo proceso. Los diseñadores de CTSS se dieron cuenta de que era más eficaz dar de una vez a los procesos con limitaciones de CPU un quantum más grande, que darles pequeños time slice con frecuencia (para reducir el intercambio). Por otro lado, si se les daba a los procesos un quantum muy grande se tendría un tiempo de respuesta pobre, como ya se ha dicho. La solución fue el establecimiento de **clases de prioridad.**

Los procesos en la clase de mayor prioridad se ejecutaban en un quantum. Los procesos de la siguiente clase se ejecutaban en dos quantums, los de la siguiente clase en cuatro quantums, etc. Cuando un proceso consumiera todos los quantums asignados a él, se le movía a la siguiente clase.

Se adoptó la siguiente política para evitar que un proceso que requería una ejecución larga al iniciar, pero que posteriormente funcionara en forma interactiva, fuera penalizado para siempre. Cada vez que se escribiera un retorno de carro (Enter) en una terminal, el proceso asociado a dicha terminal pasaba a la clase de máxima prioridad, partiendo de la hipótesis de que se convertía en interactivo. Surgió un problema con los usuarios cuyos procesos tenían serias limitaciones de uso de la CPU. Estos descubrieron que al sentarse frente a la terminal y escribir retornos de carro en forma aleatoria durante varios segundos lograron mejorar los tiempos de respuesta de sus procesos a costa de esta falencia del planificador.

Se han utilizado otros algoritmos para la asignación de procesos a clases de prioridad. Por ejemplo, el XDS 940 (Lampson, 1968), un sistema construido en Berkeley y que ejerció una gran influencia, tenía cuatro clases de prioridad, llamadas terminal, E/S, quantum corto y quantum largo. Cuando un proceso en espera de una entrada de terminal despertaba, pasaba a la clase de máxima prioridad (terminal). Cuando un proceso en espera de un bloque de disco estuviera listo, pasaba a la segunda clase. Cuando un proceso en ejecución agotaba su quantum, se le colocaba en primera instancia en la tercera clase. Sin embargo, si un proceso agotaba su quantum demasiadas veces seguidas sin bloqueo de terminal o E/S, pasaba a la cola inferior. Muchos otros sistemas utilizan algo similar para favorecer a los usuarios interactivos.

Resumen: Se particiona la cola de listos en varias colas cada una con su algoritmo de planificación.
Cada proceso pertenece a una cola y no cambia; la división se suele hacer según el tipo de proceso: batch, interactivo, procesos del sistema, etc. o por su modo de ejecución (Tiempo Real, Time Sharing, etc.).

Este algoritmo pertenece a una clase de algoritmos de planificación para situaciones en las que es fácil clasificar los procesos en diferentes grupos.

Un algoritmo de planificación con colas de múltiples nivel divide la cola de procesos listos en varias colas distintas. Los procesos se asignan permanentemente a una cola, casi siempre con base en alguna propiedad del proceso, como ser tamaño de memoria, prioridad y tipo de proceso. Cada cola tiene su propio algoritmo de planificación.

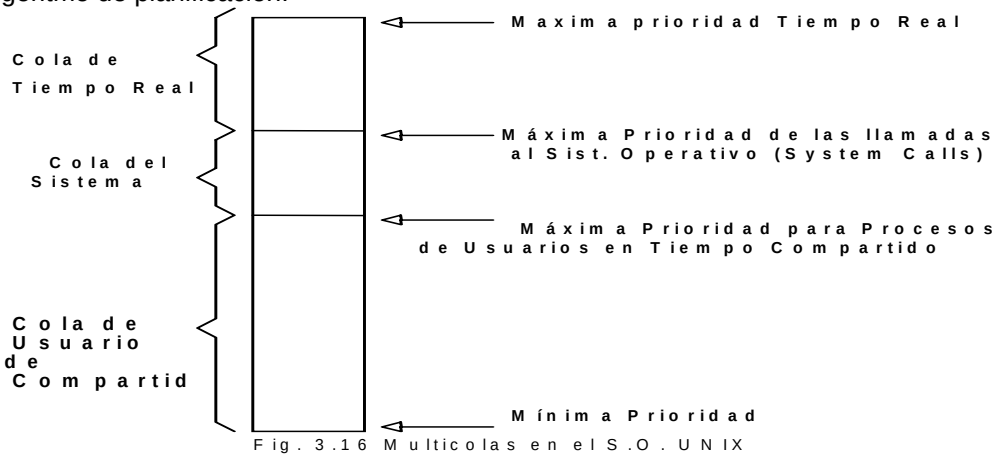


Fig. 3.16 Multicolas en el S.O. UNIX

Además debe haber planificación entre las colas, lo cual por lo regular se implementa como una planificación expropiativa (preemptive) de prioridades fijas. Por ejemplo, la cola de primer plano podría tener prioridad absoluta sobre la cola de segundo plano, por lo tanto mientras no se vacía la cola de prioridad superior, los procesos de la cola inferior no se ejecutan.

Otra posibilidad es dividir el tiempo entre las colas. Cada cola obtiene cierta proporción del tiempo de CPU, que entonces puede repartir entre los diversos procesos de su cola.

Cabe destacar que los procesos al ingresar al sistema son asignados a una cola y no pueden salir de ella, lo cual no tendría mucho sentido por la naturaleza rígida de la clasificación que realiza el algoritmo. Esto tiene la ventaja de que el gasto por planificación es bajo, y la desventaja es que es inflexible.

Formas de implementación:

1. Todos los procesos en cada cola ejecutan antes que las colas de inferior prioridad.

2. Se les da un time slice a todas las colas.
   Dentro de cada cola se puede implementar cualquier algoritmo que se desee. Ejemplo UNIX®:

3.5.5. Planificación con múltiples colas dinámicas

Es idéntico al anterior con la diferencia que los procesos se pueden mover de una cola a otra cola.
El planificador se configura usando algunos de los siguientes criterios:
1. número de colas,
2. algoritmo de planificación para cada cola,
3. método o criterio para subir/bajar un proceso,
4. criterio para determinar en que cola se pone inicialmente a un proceso.
5. es muy apropiado para esquemas client - server.

En el último Punto de éste módulo se propone varios ejemplos para el S.O. UNIX en que se pueden observar la forma de manejar las prioridades variables o dinámicas.

Para complicar más la cosa, podemos agrupar los procesos en distintas clases, y usar distintos algoritmos de planificación para cada clase. Por ejemplo, los procesos interactivos y los procesos por lotes tienen distintos requerimientos en cuanto a tiempos de respuesta. Entonces, podemos planificar los procesos interactivos usando Round Robin, y los procesos por lotes según FCFS, teniendo los primeros prioridad absoluta sobre los segundos.
Una forma de implementar este algoritmo es dividiendo la cola READY en varias colas, según la categoría del proceso. Por ejemplo, podemos tener una cola para:

- Procesos de sistema.
- Procesos interactivos.
- Procesos de los usuarios.
- Procesos por lotes.

Cada cola usa su propio algoritmo de planificación, pero se necesita un algoritmo de planificación entre las colas. Una posibilidad es prioridad absoluta con expropiación. Otra posibilidad a tener en cuenta puede ser asignarle porciones de CPU a las colas. Por ejemplo, a la cola del sistema se le puede dar el 60% de la CPU para que haga RR, a la de procesos por lotes el 5% para que asigne a sus procesos según FCFS, y a las otras el resto.
Por otra parte, podríamos hacer que los procesos migren de una cola a otra. Por ejemplo: varias colas planificadas con RR, de prioridad decreciente y quantum creciente. La última se planifica con FCFS. Un proceso en la cola *i* que no termina su fase de CPU dentro del quantum asignado, se pasa al final de la siguiente cola de menor prioridad, pero con mayor quantum. Un proceso en la cola *i* que sí termina su fase de CPU dentro del quantum asignado, se pasa al final de la siguiente cola de mayor prioridad, pero con menor quantum. Ejemplo:

Cola	0:	quantum	=	10	ms,	40%	de	CPU.
Cola	1:	quantum	=	20	ms,	30%	de	CPU.
Cola	2:	quantum	=	35	ms,	20%	de	CPU.

Cola 3: FCFS, 10% de CPU.
Así los procesos de fases más cortas tienen prioridad. Este algoritmo es uno de los más generales, pero también uno de los más complejos de implementar. También es difícil de afinar, pues hay múltiples parámetros que definir.

3.5.6. Planificación de reparto equitativo.

Todos los algoritmos de planificación expuestos hasta ahora tratan el conjunto de procesos Listos como una única reserva de procesos de donde se selecciona el que pasará a estar Ejecutando. Esta reserva puede desglosarse por prioridades pero es, normalmente, homogénea.

En un sistema multiusuario, si las aplicaciones, o los trabajos de usuarios pueden organizarse en forma de varios procesos (o hilos), se dispone de una estructura para el conjunto de procesos que no se identifica con ningún planificador tradicional.

Desde el punto de vista del usuario, el interés no está en como se comporta un proceso en particular, sino en como se comporta el conjunto de procesos de usuario que constituye una aplicación. Así pues, sería interesante poder tomar decisiones de planificación en función de estos grupos de procesos. Este método se conoce generalmente como **Planificador de reparto equitativo.**

El termino reparto equitativo hace referencia a la filosofía del planificador. Cada usuario tiene asignado algún tipo de ponderación, que indica la parte de los recursos del sistema para el usuario como una fracción de la utilización total de dichos recursos. En particular, cada usuario dispone de una parte del procesador.

3.5.7. Planificación de tres niveles (Figura 3.09)

Hasta ahora hemos supuesto de alguna manera, que todos los procesos ejecutables se encuentran en la memoria central y compiten por el uso del procesador. Si no se dispone de suficiente memoria, será necesario que algunos de los procesos ejecutables se mantengan en el disco. Esta situación tiene importantes implicaciones para la planificación, puesto que el tiempo de traer y llevar al disco los procesos para procesarlos es considerablemente mayor que el tiempo para un proceso que ya se encuentra en la memoria central. No olvidemos que las operaciones de E/S insumen varios milisegundos mientras en memoria central consumen solo algunos nanosegundos.

Una forma más práctica de trabajar con el intercambio de los procesos es por medio de un planificador de tres niveles. Primero se carga en la memoria central cierto subconjunto de los procesos ejecutables (primer nivel: long term scheduler).

El planificador se restringe entonces a ese subconjunto durante cierto tiempo. En forma periódica, se llama a un planificador de segundo nivel (middle term scheduler o swaper) para eliminar de la memoria los procesos que hayan estado inactivos demasiado tiempo. Una vez hecho el cambio, el planificador de tercer nivel (short term scheduler) restringe de nuevo a los procesos ejecutables que se encuentren en la memoria. Así, el planificador de tercer nivel se encarga de elegir de entre los procesos ejecutables que están en memoria en ese momento, mientras que el planificador de segundo nivel se encarga de desplazar los procesos de memoria a disco y viceversa.

Entre los criterios que podría utilizar el planificador de segundo nivel para tomar sus decisiones están:

¿Cuánto tiempo ha transcurrido desde el último intercambio del proceso?

- ¿Cuánto tiempo de CPU ha utilizado recientemente el proceso?
- ¿Qué tan grande es el proceso? (Los procesos pequeños no causan problemas)
- ¿Qué tan alta es la prioridad del proceso?

Aquí podríamos utilizar de nuevo round robin, planificación por prioridad o cualquiera de los demás métodos ya explicados.

3.6. Evaluación de algoritmos

La selección del algoritmo de planificación adecuado comienza por definir los criterios que se utilizarán y ordenarlos de acuerdo al perfil buscado. Una vez definido esto el siguiente paso es evaluar los diversos algoritmos que se estén considerando. Existen varios métodos de evaluación, que se describirán en las secciones siguientes.

3.6.1. Modelos determinísticos

Una clase importante de métodos de evaluación se denomina evaluación analítica. Estos métodos utilizan **el algoritmo** dado **y la carga de trabajo del sistema** para producir una fórmula o número que califica el desempeño del algoritmo para esa carga de trabajo.

Un tipo de evaluación analítica es el modelo determinista, que toma una carga de trabajo predeterminada específica y define el desempeño del algoritmo para esa carga de trabajo.

El modelo determinista es simple y rápido. Da una visión precisa permitiendo compara los algoritmos. Sin embargo, como entrada requiere números exactos y sus resultados se aplican solo a esos casos, por lo que es demasiado específico para resultar útil en la mayoría de los casos.

Un ejemplo de modelo determinista es el siguiente. Supongamos que tenemos la siguiente carga de trabajo:

Proceso	Instante de llegada	Tiempo de Ejecución
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Tabla 3.3 Carga de trabajos y tiempo de servicios de los Procesos

Los diagramas de Gantt de los distintos algoritmos nos permiten hacer un análisis del desempeño de cada uno (figura 3.17) viendo el tiempo en que finalizan la ejecución.

3.6.2. Modelo de colas

En muchos sistemas, los trabajos que se ejecutan varían diariamente, por lo que no hay un conjunto estático de trabajos (y de tiempos) como para utilizar un modelo determinístico. Sin embargo lo que si puede determinarse es la distribución de las ráfagas de CPU y de E/S. Sobre esta distribución pueden tomarse datos y luego aproximarla o estimarla. El resultado es una fórmula matemática que describe la probabilidad de una ráfaga de CPU concreta. Corrientemente se trata de una distribución exponencial, que puede describirse en términos de su media. Así mismo, debe darse la distribución de los tiempos en que los procesos llegan al sistema. A partir de estas dos distribuciones, es posible calcular el rendimiento promedio, el aprovechamiento, el tiempo de espera y demás para la mayor parte de los algoritmos.

El sistema de computación puede describirse como una red de servidores. Cada servidor tiene una cola de procesos en espera. La CPU es un servidor con su cola de listos, así como el sistema de E/S lo es de su cola de dispositivos. Si conocemos los ritmos de llegada y de servicio, podemos calcular la utilización, la longitud media de la cola, el tiempo de espera medio, etc. Esta área de estudio recibe el nombre análisis de redes de colas.

Por ejemplo, supongamos que *n* es la longitud media de cola (excluyendo el trabajo que está siendo servido), y que *W* es el tiempo medio de espera en la cola, y λ es la tasa media de llegada de nuevos procesos a la cola. Entonces podríamos esperar que durante el tiempo *W* de espera de un trabajo, llegaran a la cola $\lambda * W$ trabajos nuevos. Si el sistema está equilibrado, entonces el número de trabajos que dejan la cola tiene que ser igual al número de trabajos que llegan. Así**: *n* =** λ **** W.*** Esta ecuación se conoce como la ***fórmula de Little***, la cual es particularmente útil porque se cumple para cualquier algoritmo de planificación y distribución de las llegadas.

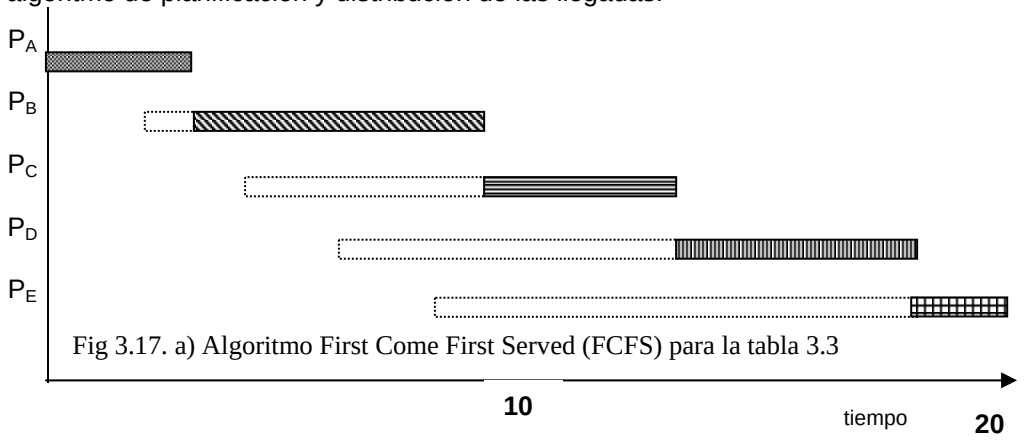


Fig 3.17. a) Algoritmo First Come First Served (FCFS) para la tabla 3.3

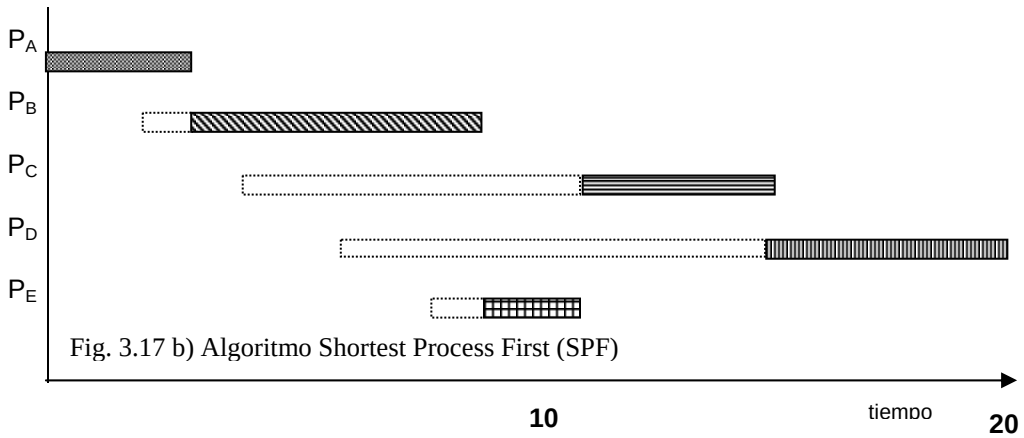


Fig. 3.17 b) Algoritmo Shortest Process First (SPF)

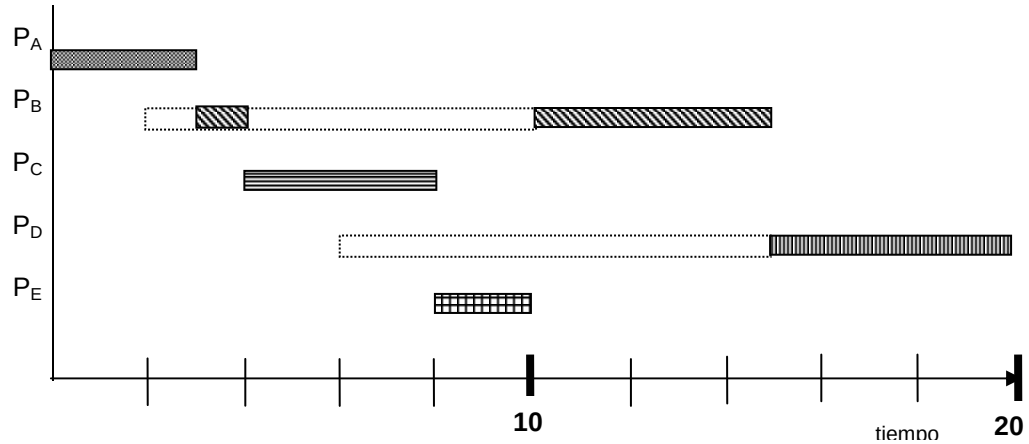


Fig 3.17. c) Algoritmo Shortest Remaning Time (SRT)

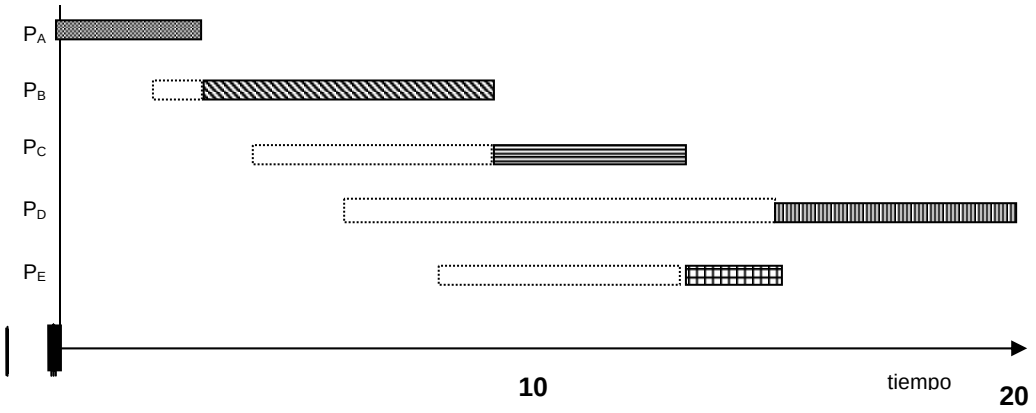


Fig 3.17. d) Algoritmo Highest Response Ratio Next (FCFS)

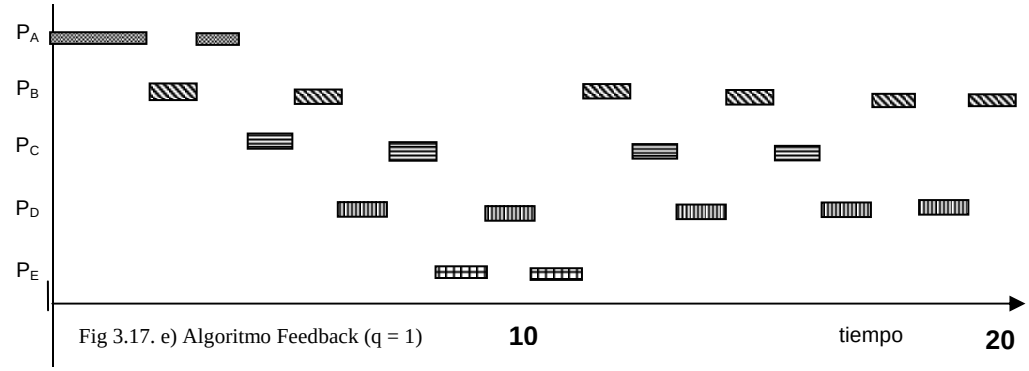


Fig 3.17. e) Algoritmo Feedback (q = 1)

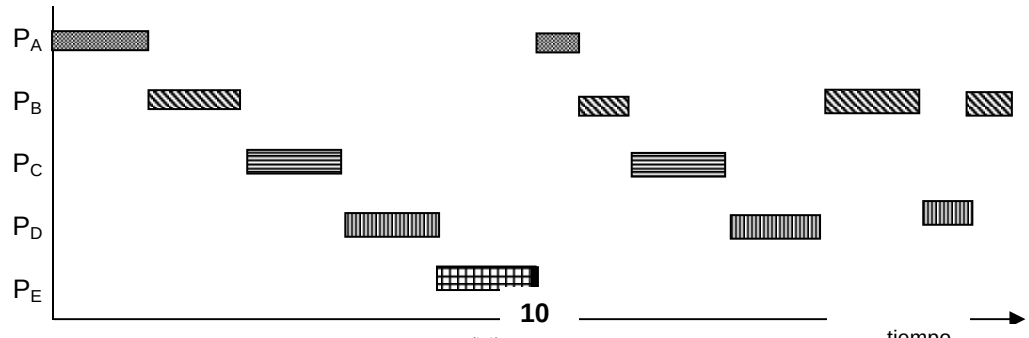


Fig 3.17. f) Algoritmo Feedback ($q = 2^{(i-1)}$)

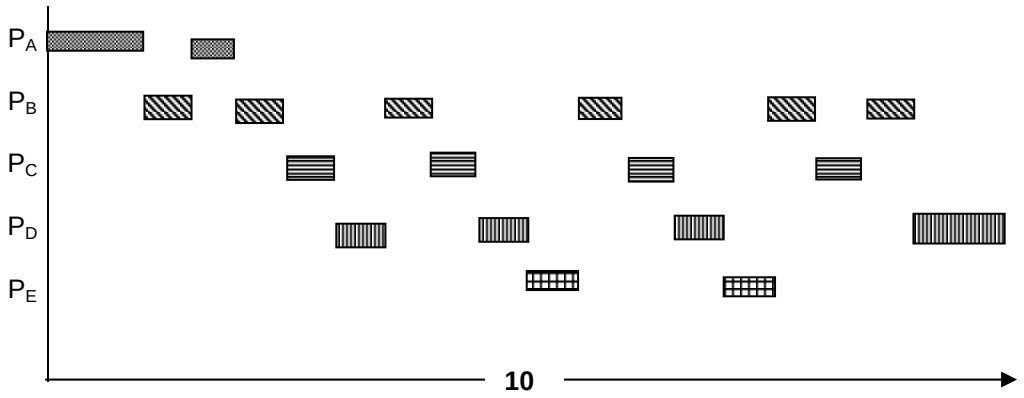


Fig 3.17. f) Algoritmo Round Robin (RR con q = 1)

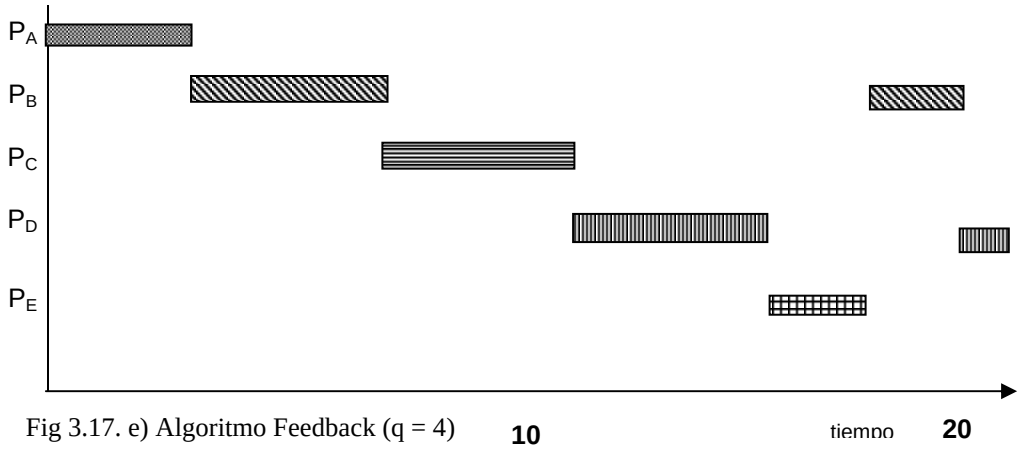


Fig 3.17. e) Algoritmo Feedback (q = 4)

Fig3.17 : Comparación de algoritmos de planificación

Aunque el análisis de colas puede ser muy útil en la comparación de algoritmos de planificación, tiene sus limitaciones. Por el momento, los tipos de algoritmos y distribuciones que pueden manejarse son bastante limitados. Puede resultar difícil trabajar con la matemática de algoritmos o distribuciones complicadas. Por tanto, las distribuciones de llegada y servicio se definen con frecuencia de modo poco realista, pero matemáticamente tratable. Así, en el cálculo de la respuesta, los modelos de las colas con frecuencia resultan solamente una aproximación al sistema real. En consecuencia, la precisión de la repuesta puede ser cuestionada.

3.6.3. Simulaciones

Se utilizan si se busca obtener una evaluación más exacta de los algoritmos de planificación. Una simulación implica programar un modelo del sistema de computación. Estructuras de datos en software representan los principales componentes del sistema. El simulador tiene una variable que representa un reloj; cuando se incrementa el valor de esta variable, el simulador modifica el estado del sistema de modo que refleje las actividades de los dispositivos, los procesos y el planificador. Conforme se ejecuta la simulación, se recopilan e imprimen datos estadísticos que indican el desempeño del algoritmo.
Las simulaciones pueden ser costosas, y a menudo requieren horas de tiempo de computador. Una simulación más detallada produce resultados más exactos, pero también requiere más tiempo de computador. Finalmente, el diseño, codificación y depuración del simulador no es una tarea trivial.
A continuación se presenta una simulación implementada en C/C++ que permite analizar varios aspectos de los algoritmos de planificación (Tabla 3.4).

Schedulingtype	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	AVG (FCFS)
Average wait time	764.39	771.72	792.16	751.28	767.74	811.24	798.48	817.95	707.26	687.27	766.949
Average arrival	23.19	24.71	21.55	23.53	22.31	23.31	22.2	24.93	20.49	23.94	23.016
Throughput	0.062383	0.067568	0.065983	0.065488	0.063532	0.060205	0.060939	0.060096	0.068181	0.06734	0.06396948
Avg time in CPU	16	14	15	15	15	16	16	16	15	14	15.2
Avg Quantum	9	8	9	9	9	9	9	9	9	8	8.8
Average Burst	16.02	14.8	15.16	15.25	15.74	16.61	16.41	16.64	15.11	14.85	15.659
Total time	1603	1480	1516	1527	1574	1661	1641	1664	1511	1485	1586.2
Idle time	1	0	0	2	0	0	0	0	0	0	0.3
Schedulingtype	SJF	SJF	SJF	SJF	SJF	SJF	SJF	SJF	SJF	SJF	AVG (SJF)
Average wait time	558.25	491.2	532.31	502.42	423.98	535.84	523.66	429.14	463.73	473.15	493.368
Average arrival	23.99	21.42	24.27	25.85	22.7	26.86	23.3	24.65	26.24	24.42	24.37
Throughput	0.05963	0.066489	0.061539	0.06398	0.070771	0.061998	0.062735	0.070274	0.06812	0.067659	0.0653193
Avg time in CPU	15	14	15	14	13	14	15	13	13	13	13.9
Avg Quantum	10	9	9	9	8	9	9	8	8	8	8.7
Average Burst	16.76	15.04	16.24	15.63	14.13	16.13	15.94	14.23	14.68	14.77	15.355
Total time	1677	1504	1625	1563	1413	1613	1594	1423	1468	1478	1535.8
Idle time	1	0	1	0	0	0	0	0	0	1	0.3
Schedulingtype	SJR	SJR	SJR	SJR	SJR	SJR	SJR	SJR	SJR	SJR	AVG (SJR)
Average wait time	480.44	493.44	574.56	491.63	504.2	455.68	519.56	474.57	513.77	482.68	493.053
Average arrival	24.44	24.64	24.35	25.35	24.7	25.49	26.06	25.22	24.87	27.19	25.23
Throughput	0.0666222	0.064267	0.060169	0.063898	0.062539	0.070028	0.063291	0.06536	0.063857	0.064851	0.06448814
Avg time in CPU	14	15	16	14	15	13	15	15	15	15	14.7
Avg Quantum	9	9	9	9	9	8	9	9	9	9	8.9
Average Burst	15.01	15.56	16.62	15.65	15.99	14.28	15.8	15.3	15.66	15.41	15.528
Total time	1501	1556	1662	1565	1599	1428	1580	1530	1566	1542	1552.9
Idle time	0	0	0	0	0	0	0	0	0	1	0.1
Schedulingtype	RR	RR	RR	RR	RR	RR	RR	RR	RR	RR	AVG (RR)
Average wait time	1080.01	1098.89	837.13	999.73	927.8	811.73	1012.49	931	958.95	922.08	957.981
Average arrival	23.42	24.44	24.63	24.22	25.53	23.67	24.83	26.62	24.55	24.75	24.666
Throughput	0.0605694	0.059685	0.069013	0.063532	0.0668	0.070721	0.062893	0.06579	0.063898	0.064809	0.06487112
Avg time in CPU	7	7	6	7	6	6	7	7	7	7	6.7
Avg Quantum	9	10	8	9	8	8	9	9	9	9	8.8
Average Burst	16.5	17.04	14.49	15.74	14.97	14.14	15.9	15.2	15.65	15.43	15.506
Total time	1651	1704	1449	1574	1497	1414	1590	1520	1565	1543	1550.7
Idle time	1	0	0	0	0	0	0	0	0	0	0.1
Schedulingtype	PR	PR	PR	PR	PR	PR	PR	PR	PR	PR	AVG (PR)
Average wait time	791.54	767.36	716.17	762.14	752.77	869.09	828.74	694.67	788.3	814.59	778.537
Average arrival	23.54	23.66	26.05	24.54	23.29	21.68	25.2	26.71	26.2	24.63	24.55
Throughput	0.0643087	0.063371	0.064893	0.06192	0.060976	0.059483	0.060241	0.06669	0.06135	0.058651	0.06220878
Avg time in CPU	15	15	15	16	16	16	16	14	15	16	15.4
Avg Quantum	9	9	9	9	9	10	9	8	9	10	9.1
Average Burst	15.55	15.78	15.4	16.15	16.4	16.81	16.6	14.94	16.3	17.05	16.098
Total time	1555	1578	1541	1615	1640	1681	1660	1495	1630	1705	1610
Idle time	0	0	1	0	0	0	0	1	0	0	0.2

Tabla 3.4: Diez simulaciones sobre algunos algoritmos de planificación

3.6.4. Implementación

Incluso las simulaciones tienen una exactitud limitada. La única forma exacta de evaluar un algoritmo de planificación es codificarlo, colocarlo en el sistema operativo, y ver cómo funciona. Este enfoque coloca el algoritmo real en el sistema real para evaluarlo en condiciones de funcionamiento reales.

El principal problema de este enfoque es el costo, ya que además de los gastos de codificación, adaptación del sistema operativo para que lo soporte, etc. existirá una reacción adversa de los usuarios, ya que estos no quieren el mejor sistema, sino que quieren que sus procesos se ejecuten y obtener sus resultados. Un sistema operativo en cambio constante no ayuda a los usuarios a hacer su trabajo.

El otro problema de cualquier evaluación de algoritmos es que el entorno en el que se usa el algoritmo cambiará. El cambio no será solo normal, a medida que se escriben nuevos programas y los tipos de problemas cambian, sino también el causado por el desempeño del planificador. Si se da prioridad a los procesos cortos, los usuarios tal vez dividan sus procesos grandes en grupos de pequeños

procesos. Si se da prioridad a los procesos interactivos sobre los no interactivos, los usuarios podrían cambiar al uso interactivo.

En la siguiente tabla presentamos como resumen de los principales algoritmos:

	FCFS	Round Robin	SPN	SRT	HRRN	feedback
Selección de Función	máx (w)	Constante	min (s)	min [s-e]	max (w+s) s	(ver texto)
Modo de Decisión	Nonpreemptive	Preemptive (según quantum)	Nonpreemptive	Preemptive (a la llegada)	Nonpreemptive	Preemptive (según quantum)
Tiempo de Respuesta	Puede ser largo	Buen tiempo en procesos cortos	Buen tiempo en procesos cortos	Buen tiempo	Buen tiempo	No enfatizado
Overhead	Mínimo	Bajo	Puede ser alto	Puede ser alto	Puede ser alto	Puede ser alto
Efectos Secundarios en los Procesos	Penaliza procesos cortos y procesos I/O Bounds	Tratamiento Libre	Penaliza procesos largos	Penaliza procesos largos	Buen balance	Puede favorecer procesos I/O Bounds

Tabla 3.5. Características de los algoritmos de planificación

3.7. Planificación de múltiples procesadores

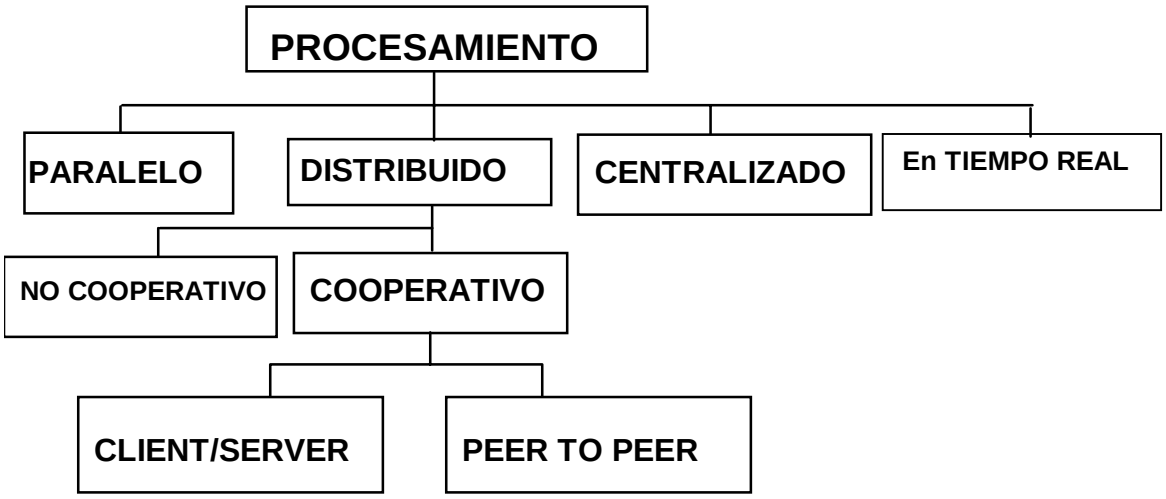


Fig. 3.18. Distintos tipos de procesamiento

Hasta ahora lo analizado se centró en los problemas de planificación en un sistema con un solo procesador. Si hay varias CPUs, el problema de planificación se vuelve más complejo.

Veamos los distintos tipos de procesamiento de datos que existen actualmente. Cada uno adopta una planificación de acuerdo a la modalidad de procesar

- **Procesamiento centralizado:** Sistema de procesamiento de datos en que las funciones de procesamiento están centralizados en una CPU y un S.O.
- **Procesamiento distribuido:** Sist. de Procesamiento descentralizado de datos que se ejecutan en nodos dispersos interconectados mediante una red.
- **Procesamiento cooperativo:** Procesamiento distribuido caracterizado por:
  - Fragmentar los elementos que componen una aplicación sobre dos o más sistemas interconectados, de igual o diferente arquitectura operativa.
  - Los recursos de los sistemas cooperantes se controlan y administran en forma independiente
  - La relación entre ambos sistemas puede tomar diferentes formas: ***Client - Server*** es un modelo de procesamiento distribuido caracterizado en que uno de los sistemas cooperante

asume el rol de cliente para solicitar un servicio específico a un proceso servidor. ***Peer to Peer*** es otro modelo en que cada nodo es igual a otro (cada uno es un par del otro). Cada uno ofrece los mismos servicios.

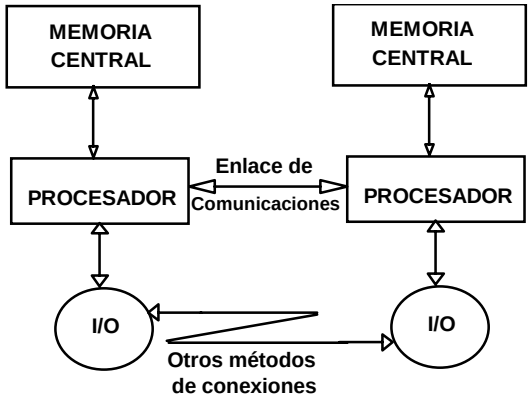


Fig 3.19 Multiprocesamiento debilmente acoplado

- **Procesamiento Paralelo:** Se caracteriza por tener múltiples procesadores trabajando sobre un espacio de memoria común. Es el caso de servidores con multiprocesadores.
- **Procesamiento en Tiempo Real:** responde a eventos producidos por fenómenos que requieren ciertas características del procesamiento como ser velocidad de respuesta, control, estadísticas, etc.

De todos estos procesamientos estudiaremos la planificación de los sistemas con multiprocesadores.

Los sistemas multiprocesadores pueden clasificarse de la siguiente manera según el uso de la memoria central:

- ***Multiprocesadores débilmente acoplados:*** consta de un conjunto de sistemas relativamente autónomos, donde cada procesador tiene su propia memoria central y sus propios canales de E/S. (caso Sist. Distribuidos o en Red)
- ***Procesadores especializados:*** similares a los procesadores de E/S. En este caso, hay un procesador principal, de propósito general; los procesadores especializados están controlados por el procesador principal y le ofrecen servicios.

- ***Multiprocesador fuertemente acoplado:*** consta un conjunto de procesadores que comparten una memoria central común y se encuentra bajo el control integrado de un sistema operativo (caso Servers)

En los Sistemas Fuertemente acoplados tienen como objetivo principal ofrecer un rendimiento mejorado y fiabilidad en la multiprogramación:

- ***Rendimiento:*** Un único multiprocesador ejecutando en un Sistema Operativo multiprogramado ofrecerá mejor rendimiento que un sistema monoprocesador equivalente y puede ser mas efectivo que varios sistemas monoprocesador.
- ***Seguridad:*** En un sistema fuertemente acoplado, si los procesadores funcionan por parejas, el fallo del procesador solo produce una degradación del rendimiento en vez de la pérdida completa del servicio.

Los procesadores débilmente acoplados es el caso del procesamiento distribuido. Se acopla a través de CPU o I/O. Las Configuraciones híbridas y algunos Sistemas débilmente acoplados permiten a los procesadores acceder a una memoria no local e incluso a la memoria privada de otros procesadores. Existen retardos provocados por el arbitraje de accesos y el ruteo de interconexiones entre procesador y memoria.

Otra clasificación de procesadores según el método de acceso a la memoria:

- **UMA (uniform memory access):** Sistema en que todos los procesadores pueden acceder a toda la memoria disponible con la misma velocidad (de forma uniforme). Ejemplo: BUS compartido.·

- **NUMA (non uniform memory access):** Sistema en que hay diferencias de tiempo en el acceso a diferentes áreas de Memoria. El procesador accede de una forma no uniforme. Depende de la proximidad del procesador y de la complejidad del mecanismo de conmutación.·
- **NORMA (non remote memory access):** no existen memorias compartidas.

Las Arquitecturas de computadoras que utilizan multiprocesadores son: a) Orientado a bus, b)Crosstalk.(barras cruzadas), c) Hipercubos y d) Conmutadores múltiples.

Todas estas arquitecturas pueden tener pocos procesadores o muchos (depende de la granularidad). Cada procesador puede ejecutar un proceso o varios simultáneos. Cada proceso puede estar ejecutando un Hilo o muchos. Esto se puede generar conflictos con la planificación. De ahí que no es sencillo determinar una política de planificación.

En la planificación de multiprocesadores antes de aplicar algún método, se deben tener en cuenta estos tres puntos (que a su vez están interrelacionados):

*La asignación de procesos a procesadores.
*El uso de multiprogramación en los procesadores individuales
*La expedición real de los procesos

El punto de asignación de procesos se refiere a que si se supone que la arquitectura de multiprocesadores es **uniforme o homogéneo** (todos los procesadores son iguales), el método más simple consiste en tratar a los procesadores como un recurso reservado, y asignar los procesos a los procesadores por demanda. Esta asignación se hace de forma estática (desde su activación hasta su terminación debe mantenerse una cola a corto plazo para cada procesador) en donde encontramos la ventaja de que el costo es menor porque la asignación del procesador se realiza una vez y para siempre, pero también cuenta con algunas desventajas que pueden ser solucionadas por el programador, como ser que un procesador puede estar desocupado con su cola vacía, mientras que otro tiene muchos trabajos pendientes. También podemos realizar la asignación de forma dinámica.

Con un acoplamiento muy fuerte y una arquitectura de memoria compartida, la información de contexto de todos los procesadores se encuentra disponible para todos los procesadores y por lo tanto el costo será independiente de la identidad del procesador.

También encontramos otras formas de asignar procesos a los procesadores para esto se pueden utilizar dos métodos, la arquitectura maestro esclavo y las arquitecturas simétricas (SMP Simetric Multi Processing), por supuestos hay un amplio abanico de soluciones entre estos dos extremos, unas consisten en que un subconjunto de los procesadores se dediquen a ejecutar el núcleo en lugar de hacerlo uno solo y el otro método solo consiste en controlar las diferencias entre las necesidades de los procesos del Kernel y de otros procesos en función de prioridades. En el segundo es en el que se encuentran varios procesadores disponibles, no es necesario que cada procesador este ocupado al máximo, sino que haya que tratar de obtener el mejor rendimiento en promedio para las aplicaciones.

En él ultimo punto (expedición de un proceso) se dice que esta orientado en la selección real del proceso a ejecutar. En un sistema monoprocesador multiprogramado el uso de prioridades o algoritmos de planificación sofisticados nos llevan al mejoramiento de la estrategia FIFO en cambio cuando se habla de multiprocesadores esta complejidad puede ser innecesaria y contraproducente.

Cuando hay varias CPUs (y una memoria común), la planificación también se hace más compleja. Podríamos asignar una cola READY a cada procesador, pero se corre el riesgo de que la carga quede desbalanceada: algunos procesadores pueden llegar a tener una cola muy larga de procesos para ejecutar, mientras otros están desocupados (con la cola vacía). Para prevenir esta situación se usa una cola común de procesos listos, para lo cual hay dos opciones:

1. Cada procesador es responsable de su planificación, y saca procesos de la cola READY para ejecutar. ¿El problema? Hay ineficiencias por la necesaria sincronización entre los procesadores para acceder a la cola.
2. Dejar que sólo uno de los procesadores planifique y decida qué procesos deben correr los demás: **multiprocesamiento asimétrico**.

3.7.1. Granularidad

Una buena forma de caracterizar los multiprocesadores y situarlos en el contexto de otras arquitecturas es considerar la granularidad de la sincronización o frecuencia de sincronización entre los procesos de un sistema. Es posible distinguir cuatro categorías de paralelismo que difieren en el grado de granularidad:

- Paralelismo independiente,
- Paralelismo de grano grueso y muy grueso,

- Paralelismo de grano medio,
- Paralelismo de grano fino.

Con el paralelismo independiente, no existe sincronización explícita entre los procesos. Cada uno representa una aplicación o trabajo separado e independiente. Un uso clásico de este tipo de paralelismo se da en sistemas de tiempo compartido. Cada usuario esta ejecutando una aplicación en particular, como un procesador de texto o una hoja de cálculo. Los multiprocesadores ofrecen el mismo servicio que un monoprocesador multiprogramado**.** Esto vale para la planificación de procesos y usuarios.

Con el paralelismo de grano grueso y muy grueso, existe una sincronización entre los procesos pero a un nivel muy burdo. Este tipo de situación se maneja fácilmente con un conjunto de procesos concurrentes ejecutando en un monoprocesador multiprogramado y puede verse respaldado por multiprocesadores con escasos cambios o incluso ninguno en el software del usuario.

En el caso de grano medio, el programador debe especificar explícitamente el posible paralelismo de la aplicación. Normalmente, hará falta un grado más alto de coordinación e interacción entre los hilos de la aplicación, lo que lleva a un nivel de sincronización de grano medio. Puesto que los diversos hilos de una aplicación interactúan de forma muy frecuente, las decisiones de planificación que involucren a un hilo pueden afectar el rendimiento de la aplicación completa.

El paralelismo de grano fino significa un uso del paralelismo mucho más complejo que el que se consigue con el uso de hilos. Si bien gran parte del trabajo se realiza en aplicaciones muy paralelas, este es un campo, hasta el momento, muy especializado y fragmentado, con varias soluciones diferentes.

Existen básicamente dos tipos de sistemas multiprocesadores, los homogéneos, donde los procesadores tiene características similares o iguales y los heterogéneos.

En los sistemas heterogéneos cada procesador se encargará de ejecutar los procesos específicamente compilados para él. Los sistemas homogéneos también cuentan con limitaciones para la planificación, ya que si por ejemplo un dispositivo se encuentra en el bus privado de un procesador todos los procesos que necesiten hacer uso del mismo deberán colocarse en su cola de listos.

A partir de ahora sólo se tendrán en cuenta los sistemas que cuentan con un conjunto de procesadores que comparten una memoria común y están completamente bajo el control del sistema operativo.

Tamaño Grano	Descripción	Intervalo de sincronización (Instrucciones)
Fino	Paralelismo inherente en un único flujo de instrucciones	< 20
Medio	Procesamiento paralelo o multitarea dentro de una aplicación individual	20 - 200
Grueso	Multiprocesamiento de procesos concurrentes en un entorno multiprogramado	200 - 2000
Muy Grueso	Proceso distribuido por los nodos de una red para formar un solo entorno de computación	2000 – 1M
Independiente	Varios procesos no relacionados	(N/A)

Tabla 3.6. Granularidad

Elementos de diseño: Si se cuenta con varios procesadores idénticos, puede compartirse la carga. Seria posible establecer una cola aparte para cada procesador, aunque esto permitiría que un procesador este ocioso, mientras otros están sobrecargados. Para solucionar esto se implementa una cola de **procesos listos única**, en este caso todos los procesos ingresan en una cola y se le asigna cualquier procesador disponible. A esto se lo denomina **asignación dinámica**.

En un esquema así, podría usarse una de dos estrategias de planificación. En la primera, cada procesador se auto planifica. El procesador examina la cola común y escoge el proceso que ejecutará. Esto debe ser programado con mucho cuidado ya que dos procesadores no pueden leer al mismo tiempo una estructura de datos (la cola de listos), por lo que se corre el riesgo de que dos procesadores escojan el mismo proceso o de perder procesos. La otra estrategia consiste un nombrar a un procesador como planificador de los demás, lo que da lugar a una estructura maestro-esclavo.

Algunos sistemas llevan esta estructura un paso más allá y hacen que un procesador, el servidor maestro, tome todas las decisiones de planificación y se encargue del procesamiento de E/S y otras actividades del sistema, mientras los demás procesadores se encargan de ejecutar el código de usuario. Este **multiprocesamiento asimétrico** es mucho más sencillo que el simétrico, porque solo un procesador accede a las estructuras de datos del sistema y se reduce la necesidad de compartir datos y de controlarlos.

Si la asignación es estática, surge la pregunta de si puede estar multiprogramado el procesador. En los multiprocesadores tradicionales con sincronización de grano grueso o independiente cada procesador individual podría alternar entre varios procesos para conseguir una alta utilización y por tanto un buen rendimiento. Cuando se trabaja con aplicaciones de grano medio, se debe tratar de obtener el mejor rendimiento, en promedio, para las aplicaciones. Una aplicación que conste de varios hilos puede rendir poco a menos que todos sus hilos estén disponibles para ejecutar simultáneamente.

Además se deben tener en cuenta los algoritmos utilizados para el despacho de procesos, ya que un algoritmo complejo, como los utilizados en sistemas monoprocesador, puede ser innecesaria e incluso contraproducente, por lo que la utilización de una técnica sencilla puede ser más efectiva y aportar menos sobrecarga.

3.7.2. Planificación de procesos

En la mayoría de los sistemas multiprocesadores homogéneos tradicionales, los procesos no se asignan a los procesadores de forma dedicada ya que en su lugar hay una cola única para todos ellos, o si es un esquema de prioridades existen varias colas según la prioridad que van todas a una reserva común de procesadores, pero en cualquiera de estos dos casos es posible contemplar al sistema como una arquitectura de colas multiservidor.

En el caso de tener un sistema dual cada procesador tiene la mitad de taza de un proceso, y se

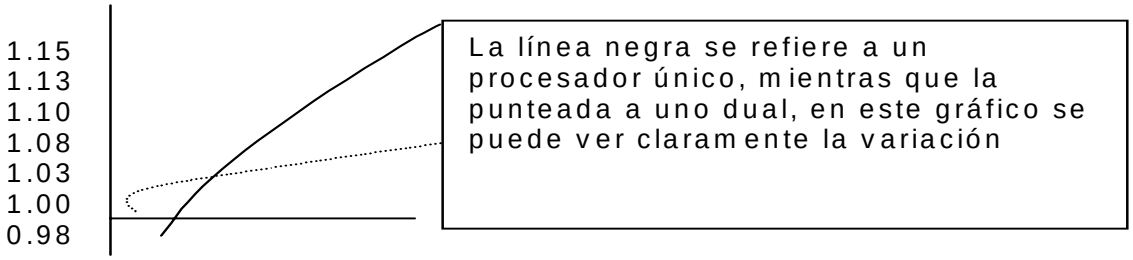


Fig. 3.20. Comparación de Sistemas Monoprocesador y Dual.

realiza un análisis de colas que compara la planificación de FCFS con un turno rotatorio y con el algoritmo de menor tiempo restante, en donde sus resultados. Pueden variar según como se produzcan los tiempos de servicios.

3.7.3. Planificación de hilos

Los hilos son cada vez más utilizados en los S.O. actuales y los lenguajes. Con el uso de estos hilos el concepto de la ejecución se separa del resto de la definición de un proceso ya que una aplicación puede implementarse como un conjunto de hilos que cooperan y ejecutan concurrentemente en el mismo espacio de direcciones. En un sistema monoprocesador los hilos pueden utilizarse para ayudar a la estructuración del programa, y suponer la E/S y el procesamiento superpuestos. Puesto que el tiempo utilizado en el intercambio de hilos es menor al comparado con el de procesos, sin embargo los beneficios de los hilos se notan más en un sistema multiprocesador ya que en estos se pueden emplear hilos para obtener un paralelismo en la ejecución de las aplicaciones.

La potencia de los hilos se hace evidente en un sistema multiprocesador donde pueden emplearse para obtener un paralelismo real en las aplicaciones. Si los diversos hilos de una aplicación se ejecutan simultáneamente en distintos procesadores, se posibilita un aumento drástico del rendimiento. Sin embargo, puede demostrarse que en aplicaciones que exijan una interacción considerable entre los hilos, pequeñas diferencias en la planificación y gestión de hilos pueden tener un importante significado en el rendimiento. Entre las diversas propuestas en la planificación de hilos de multiprocesadores y de asignación de procesos se destacan los siguientes métodos:

1. **Compartir carga (Load sharing):** los procesos no se asignan a un procesador en particular. Esto es válido para procesadores homogéneos. Se mantiene una cola global de hilos listos y cada procesador, cuando está ocioso, selecciona un hilo de la cola. Las ventajas de esta técnica

son que la carga se distribuye uniformemente entre los procesadores, que no es necesario un planificador centralizado y que la cola global puede organizarse con alguno de los esquemas de monoprocesadores. Hay tres versiones diferentes de compartir la carga: FCFS, primero el de menor número de hilos y primero el de menor número de hilos con expropiación. Las desventajas de esta técnica son: que la cola central ocupa una región de memoria a la que se debe acceder con mutua exclusión, que es improbable que los hilos expulsados reanuden su ejecución en el mismo procesador y que es improbable que todos los hilos de un programa consigan acceder a los procesadores al mismo tiempo.

2. **Planificación por grupos (gang scheduling):** se planifica un conjunto de hilos afines para su ejecución en un conjunto de procesadores al mismo tiempo, según una relación uno a uno. Las ventajas de esta técnica son que: si procesos relativamente próximos se ejecutan en paralelo, pueden reducirse los bloqueos por sincronización, pueden hacer falta menos intercambios de procesos y se incrementará el rendimiento; y la sobrecarga de planificación puede reducirse. Este tipo de planificación es necesario para aplicaciones paralelas de grano medio a fino cuyo rendimiento se degrada cuando alguna parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. El empleo de planificación por grupos origina un requisito de asignación del procesador. Una posibilidad es la siguiente: supóngase que se tienen N procesadores y M aplicaciones, cada una tiene N hilos o menos, si se fracciona el tiempo, cada aplicación podría tener 1/M del tiempo disponible de los N procesadores. Este reparto uniforme puede generar un derroche de los recursos del procesamiento. Como alternativa se puede utilizar la planificación ponderada por el número de hilos, donde cada aplicación obtiene una parte del tiempo relacionada con la cantidad de hilos que posee.

3. **Asignación dedicada de procesadores:** ofrece una planificación implícita definida por la asignación de hilos a los procesadores. Se usa para procesadores heterogéneos. En el tiempo que se ejecuta un programa, a cada programa se le asigna un número de procesadores igual al número de hilos que posea. Cuando el programa finaliza, los procesadores retornan a la reserva general para posibles asignaciones a otros programas. Las ventajas de esta estrategia son: en un procesador masivamente paralelo, con muchísimos procesadores, cada uno de los cuales representa una pequeña parte del costo del sistema, la utilización del procesador no es tan importante como medida de la efectividad; y la anulación total del intercambio de procesos durante el tiempo de vida de un programa dará como resultado una aceleración sustancial del programa. La clave está en cómo se asignan muchos procesadores a un programa en un instante dado, uno de los conceptos que se utiliza es el de conjunto de trabajo de actividades, que es el número mínimo de actividades (hilos) que deben ser planificadas simultáneamente en los procesadores para que la aplicación tenga un progreso aceptable. Un fallo en este tipo de planificación puede conducir a una hiperplanificación del procesador. A su vez la fragmentación del procesador se refiere a la situación en la que sobran algunos procesadores mientras otros están asignados y los procesadores sobrantes son insuficientes en número o están mal organizados.

4. **Planificación dinámica:** el número de hilos en un programa puede cambiar en el curso de una ejecución. El S.O. es responsable de repartir los procesadores entre los trabajos. Cada trabajo emplea los procesadores de su partición para procesar un subconjunto de sus tareas ejecutables, organizando estas tareas en hilos. A las aplicaciones individuales se les deja la decisión sobre el subconjunto a ejecutar, además de a que hilo suspender cuando se expulsa un proceso. Con este método, la responsabilidad de planificación del S.O. se limita sobre todo a la asignación del procesador. En primer lugar se asignarán los procesadores desocupados, en segundo se le quitará un procesador a algún proceso que tenga más de un asignado, sino la petición quedará pendiente.

3.8. Planificación en tiempo real

El SO y el planificador es el componente más importante de un S.O. de tiempo real, estos cuentan con características diferenciales en las siguientes cinco áreas:

- **Determinismo:** en un S.O. de tiempo real, las solicitudes de los procesos vienen dictadas por sucesos y determinaciones externas, el punto en el cual el S.O. puede satisfacer las peticiones

de forma determinista depende en primer lugar de la **velocidad** con que pueda responder frente a las interrupciones, y a la **capacidad** para gestionar todas las peticiones en el tiempo exigido.

- **Sensibilidad**: tanto el determinismo como la sensibilidad forman parte del tiempo de respuesta a sucesos externos, en este tipo de sistema estos requisitos de tiempo son críticos ya que deben actuar con tiempos impuestos por individuos, dispositivos, y flujos externos al sistema
- **Control de usuario:** en un S.O. de tiempo real resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas, también debe permitirle especificar características de paginación o intercambio de procesos
- **Fiabilidad:** este punto es muy importante en los sistemas de tiempo real ya que las pérdidas o degradaciones del rendimiento pueden causar graves daños desde financieros hasta daños en equipos o del procesamiento.
- **Tolerancia a fallos:** un S.O. de tiempo real debe ser diseñado para responder incluso a varias formas de fallo, intentando corregirlo o minimizándolo para poder seguir con la ejecución. Este tipo de sistemas debe ser estable.

Los sistemas de tiempo real se programan con frecuencia como colección de pequeñas tareas, cada una con funciones bien definidas y un tiempo de ejecución con cota también bien definida. La respuesta a un estímulo dado puede requerir la ejecución de varias tareas, por lo general con restricciones acerca de su orden de ejecución. Además hay que tomar una decisión en relación con las tareas por ejecutar en tal o cual procesador.

En cuanto a la planificación se utilizan distintos métodos ya que estos dependen de: a) si el sistema lleva a cabo un análisis de la planificación, en caso de que lo llevara si se realiza estática o dinámicamente, b) si el resultado del análisis genera un plan con respecto al cual se expiden las tareas durante la ejecución, a partir de estos puntos se identifican diversas clases de algoritmos utilizados para la planificación:

- **Métodos con tablas estáticas:** se realiza un análisis estático de las planificaciones posibles que da como resultado cuando se debe comenzar la realización de una tarea. Se aplica a tareas periódicas, la entrada del análisis consta con el tiempo de llegada, el tiempo de ejecución, el plazo de finalización y la prioridad de cada tarea. El planificador intenta trazar un plan el cual le permita cumplir con todas las tareas, el problema de esta estrategia se da cuando se necesita cambiar alguna tarea, ya que luego se debe trazar de nuevo todo el plan.
- **Métodos expropiativos con prioridades estáticas:** también se realiza un análisis pero en este caso se usa para poder asignar prioridades a las tareas. Cuando hablamos de sistemas de tiempo real aplicamos esta estrategia para la asignación de prioridades que se encuentran relacionadas con todas las restricciones de tiempo asociadas a cada tarea.
- **Métodos dinámicos de planificación:** este algoritmo funciona aceptando una nueva tarea para ejecutar solo si es factible cumplir con sus restricciones de tiempo. Uno de los resultados que otorga este análisis es el orden en que debe ser expedida la tarea. Este algoritmo no realiza ningún tipo de análisis de viabilidad, el sistema no intenta cumplir con todos los plazos y abandona cualquier proceso ya iniciado y cuyo plazo no se haya cumplido.

La planificación puede ser empleada adaptando diferentes estrategias siguiendo siempre con este concepto de estático y dinámico.

Planificación dinámica: se utiliza después de que llego la tarea, pero antes de ejecutarla y se crea un plan que contenga todas las que estaban previamente planificadas.

Planificación dinámica del mejor resultado: esta técnica es la que mayormente se utiliza en los sistemas de tiempo real, funciona del siguiente modo: al llegar una tarea el sistema en función de sus características le asigna una prioridad, su ventaja está en la facilidad de implementación ya que el modo estático no siempre es el mas conveniente porque en general las tareas son periódicas y no es posible un análisis estático de su planificación.

Los algoritmos de planificación de tiempo real se pueden caracterizar mediante los siguientes parámetros:

- Tiempo real duro vs. Tiempo real suave o blando.
- Planificación con prioridad vs. Sin prioridad.
- Dinámico vs. Estático.
- Centralizado vs. Descentralizado.

Los **algoritmos de tiempo real duro** deben garantizar que se cumple con los tiempos límite. Los algoritmos de tiempo real suave pueden vivir con un método del mejor esfuerzo. El caso más importante es el tiempo real duro.

La **planificación con prioridad** permite suspender una tarea de manera temporal al llegar una con mayor prioridad, y continuar su ejecución cuando no existen más tareas de mayor prioridad por ejecutar. La planificación sin prioridad ejecuta cada tarea hasta terminarla. Una vez que se inicia una tarea, continúa conservando a su procesador hasta terminarla. Se utilizan ambos tipos de estrategias de planificación.

Los **algoritmos dinámicos** toman las decisiones de planificación durante su ejecución. Al detectar un evento, un algoritmo dinámico con prioridad decide en ese momento si ejecuta la primera tarea asociada con el evento o continua ejecutando la tarea activa. Un algoritmo dinámico sin prioridad sólo observa que otra tarea se puede ejecutar. Cuando termina la tarea activa, elige entre las tareas listas para su ejecución.

Por el contrario, con los **algoritmos estáticos**, las decisiones de planificación, con prioridad o sin ella, se toman de antemano, antes de la ejecución. Cuando ocurre un evento, el planificador del tiempo sólo observa una tabla para ver que hacer.

Por último, la planificación puede ser centralizada, de modo que una máquina recoge toda la información y toma todas las decisiones, o puede ser descentralizada, de forma que cada procesador tome sus decisiones. En el caso centralizado, la asignación de tareas a los procesadores se puede realizar en el mismo momento. En el caso de la descentralizada, la asignación de tareas a los procesadores es distinta de la decisión del orden de ejecución de las tareas asignadas a un procesador dado.

Como consecuencia de la independencia entre tareas, puede asumirse que en algún instante de tiempo (**instante crítico)** las tareas serán liberadas juntas.

Los métodos de planificación

Inicialmente consideraremos un modelo de tareas muy simple para describir algunos métodos estándar de planificación. Tiene las siguientes características:

– La aplicación está formada por un conjunto fijo de tareas.
– Todas las tareas son periódicas, con periodos conocidos.
– Las tareas son completamente independientes entre sí.
– Todas las sobrecargas del sistema, duración de los cambios de contexto, etc., son ignoradas (se asume que tienen coste cero).
– Cada tarea tiene un plazo igual a su periodo.
– Todas las tareas tienen un tiempo de ejecución máximo conocido.

Como consecuencia de la independencia entre tareas, puede asumirse que en algún instante de tiempo (**instante crítico)** las tareas serán liberadas juntas.

Los Parámetros de planificación en tiempo real a tener en cuenta son:

- *N* : Número de tareas.
- *T* : Periodo de activación.
- *D* : Plazo de respuesta
- *C* : Tiempo de ejecución máximo.
- *R* : Tiempo de respuesta máximo.
- *P* : Prioridad.

Un **método de planificación puede** presentar dos aspectos importantes:

1. Un algoritmo de planificación que ordena el uso de los recursos del sistema (en particular, la CPU).
2. Un método de análisis que permita predecir el comportamiento temporal del sistema en el peor caso cuando se aplica el algoritmo de planificación.

Características de las políticas de planificación

Los objetivos que persigue toda política de planificación de tiempo real son:

- Garantizar la correcta ejecución de todas las tareas críticas.
- Ofrecer un buen tiempo de respuesta a las tareas aperiódicas sin plazo.
- Administrar el uso de recursos compartidos.
- Posibilidad de recuperación ante fallos de software o hardware.
- Soportar cambios de modo.

- Cambiar en tiempo de ejecución el conjunto de tareas. Por ejemplo: un cohete espacial tiene que realizar acciones muy distintas durante el lanzamiento, estancia en órbita y regreso; en cada fase, el conjunto de tareas que se tengan que ejecutar ha de ser distinto.

3.8.1. Planificación dinámica

Estos son algunos de los algoritmos de planificación dinámica más conocidos:

- **Algoritmo monótono de tasa:** Fue diseñado para tareas de planificación con prioridad, sin restricciones de orden o exclusión mutua en un procesador. De antemano, cada tarea tiene asignada una prioridad igual a su frecuencia de ejecución. Al tiempo de ejecución, el planificador siempre selecciona la tarea de máxima prioridad para su ejecución, priorizando sobre la tarea activa en caso de ser necesario.
- **Algoritmo del primer límite en primer lugar:** Siempre que se detecta un evento el planificador lo agrega a la lista de tareas, la cual se va ordenando por tiempo límite. El planificador entonces sólo elige la primera tarea de la lista, la más cercana a su tiempo límite.
- **Laxitud:** Este algoritmo calcula primero el tiempo total ocupado por cada tarea. Una vez que se tiene este valor lo resta al tiempo asignado para la tarea. De esta forma la cola se va ordenando de acuerdo a la laxitud o "espacio para respirar".

Ninguno de estos algoritmos han demostrado ser óptimos en un sistema distribuido. Además, ninguno de ellos toma en cuenta las restricciones de orden o de mutua exclusión, incluso en un monoprocesador.

3.8.2. Planificación estática

La planificación estática se realiza antes de que el sistema comience su operación. La entrada consiste en una lista de todas las tareas que se deben ejecutar.

Lo importante de esta planificación es que el comportamiento al tiempo de ejecución es por completo determinista, y se conoce antes de que el programa inicie su ejecución, lo que garantiza que se cumplirán los tiempos límites, salvo que existan errores de comunicación o de procesadores. Los errores de CPU igualmente se pueden salvar haciendo que dos o más procesadores se examinen de manera activa entre sí.

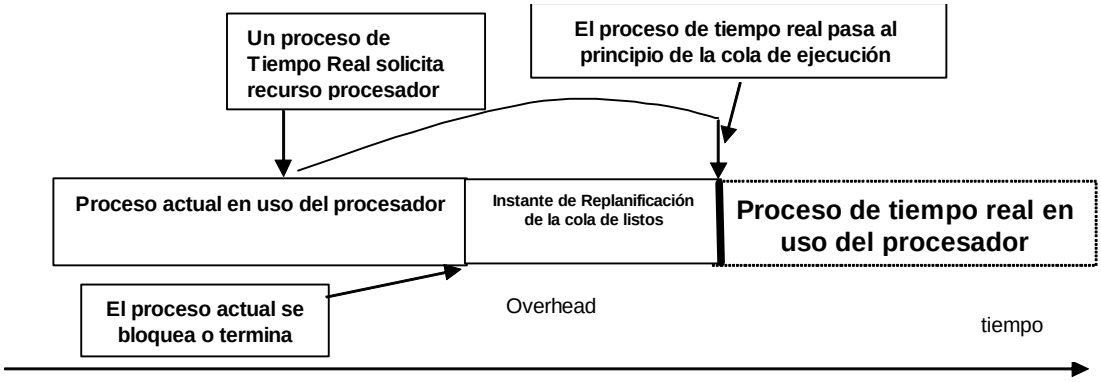


Fig. 3.21. Llegada de un proceso en Tiempo Real y su planificación.

La planificación puede realizarse de las siguientes formas:

- **Planificación con tablas estáticas:** es aplicable a tareas periódicas. El planificador intenta trazar un plan que le permita cumplir las exigencias de todas las tareas periódicas.
- **Planificación expropiativa con prioridades estáticas:** hace uso del mecanismo de planificación no apropiativo con prioridades. En un sistema de tiempo real, la asignación de prioridades se encuentra relacionada con las restricciones de tiempo asociadas a cada tarea.

3.8.3. Planificación por plazos

Han aparecido muchos métodos potentes y apropiados para la planificación de tareas de tiempo real, todos ellos se basan el en hecho de poseer información adicional sobre todas las tareas, en realidad seria necesario disponer de**: el instante en que esta lista la tarea, plazo de comienzo, plazo de finalización, tiempo de proceso, exigencias de recursos, prioridad y estructuras asociadas.**

Cuando se consideran plazos hay varios factores a tener en cuenta en la planificación en tiempo real, como por ejemplo que tarea se planifica a continuación y que tipo de expropiación se permite. Otro punto critico en el diseño es la expropiación, con esto nos referimos a cuando se especifican los plazos de comienzo, usar un planificador expropiativo, en el cual es propio de la tarea de tiempo real el tener que bloquearse así misma después de completada su parte critica de ejecución permitiendo así satisfacer otros plazos de tiempo real.

Planificación monótona en frecuencia (RMS - Rate Monotonic Schedulling)

Es la solución para la planificación multitarea con tareas periódicas, en RMS la tarea de mas alta prioridad es la de periodo mas corto, la segunda tarea de mayor prioridad es la de periodo mas corto así sucesivamente, si se tienen para ejecutar mas de una tarea se atiende primero a la que posea el periodo mas corto, el siguiente gráfico ilustra la prioridad de las tareas en función de sus frecuencias.

Uno de los beneficios mas importantes que ofrece RMS es la estabilidad ya que este estructura las tareas fundamentales para que tengan periodos cortos o modificando las prioridades de RMS para que tengan en cuenta a dichas tareas.

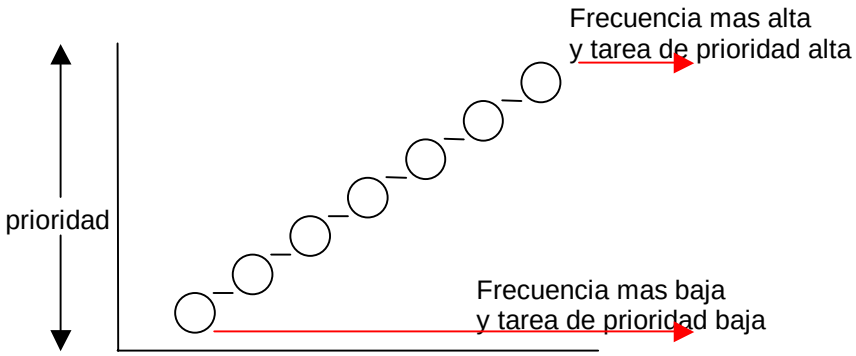


Fig. 3.22. Planificación en Tiempo Real Monótona en frecuencia.

- Período T: tiempo que transcurre entre una llegada de la tarea y la siguiente llegada de la misma tarea.
- Frecuencia F: es la inversa del período.
- Tiempo de ejecución C: tiempo de procesamiento de cada acontecimiento de una tarea, que en un sistema monoprocesador cumple C<=T, y la utilización por parte de la tarea si es que nunca se le niega el servicio es U= C/T.

3.9 Bibliografía recomendada para este Módulo

Operating Systems. (Fourth Edition), Internals and Design Principles, Stallings Willams, Prentice Hall, Englewood Cliff, NJ., 2001,

Operating Systems Concepts Fiftth Edition, Silberschatz, A. and Galvin P. B., Addison Wesley, 1998,

Operating Systems Concepts and Design (Second Edition), Milenkovic, Millan., Mc Graw Hill, 1992

Modern Operating Systems, Tanenbaum, Andrew S., Prentice - Hall International, 1992

Operating Systems, Design and Implementation (Second Edition), Tanenbaum, Andrew S., Prentice - Hall International, 1997

Fundamentals of Operating Systems., Lister, A.M. , Macmillan, London; 1979

Operating System Design - The XINU Approach; Comer, Douglas; Prentice may; 1984

Operating System; Lorin, H., Deitel, H.M.; Addison Wesley; 1981

The UNIX Operating System; Christian, K.; John Wiley; 1983

UNIX System Architecture; Andleigh, Prabhat K..; Prentice - Hall International; 1990

The UNIX Programming Environment; Kernighan, B.W. and Pike, R.; Prentice - Hall International; 1984

The UNIX System V Environment; Bourne, Stephen R.; Addison Wesley; 1987

Sistemas Operativos Conceptos y diseños.(Segunda Edición); Milenkovic Milan.; Mc Graw Hill; 1994..

Sistemas de Explotación de Computadores; CROCUS; Paraninfo; 1987 424 pág.

GLOSARIO DE TÉRMINOS EN IDIOMA INGLÉS

Users	Thrashing	Scheduler
Ready Queue	Job	Job Scheduler
Long term Scheduler	Macro Scheduler	High level Scheduler
Low level Scheduler	Low Scheduler	Process Scheduler
Meddle term Scheduler	Short Term Scheduler	Medium term Scheduler
Batch	Multiaccess	Peopleware
Orgware	Ready Queue	Back up
Suspend Queue	Bloqued Queue	Completed Queue
Waiting	Signal	Process
Spooler	Create Process	Initiate
Running	Select	Terminate
Dispatcher	Handler	Delay Process
Time Sharing	Traffic Controller	Switcher
Kernel	Overhead	Context switch
End	Top	Preemptive
Non Preemptive	Running state	Wait state
Wait for I/O state	Wait state	Ready state
terminate state	I/O Burst	CPU Burst
I/O Bound	CPU Bound	Blocked I/O
Throughput	turnaround time	Locked I/O
Shortest Job Next	Shortest Job First	First Come First Served
Shortest Remaining Time First	Starvation	Aging
Quantum	Time slice	Round Robin
System calls	computed time	Elapsed Time
Turnaround time	waiting time	Response time
Time Slice	Throughput time	Response tieme
Default	Running time	Time out
The running process	Clock Tick	System Privilege
Switch in	Switch out	Run for ever
Switch out	Terminal I/O	Disk I/O
Swaping	Swap out	Swap in
Hash	Current Process	Buffer address
Tereminal input priority	Hashing	Terminal address
Switcher	Wakeup	Syscall
Disk I/O Request	Sleep	Pointer
Dispatch Latency	Lowest High Priority	System Flag
Crosstalk	Cooperative multitasking	Peer to Peer

GLOSARIO DE TÉRMINOS EN CASTELLANO

Planificador	Algoritmo de planificación	Niveles de planificación

Planificación extra largo plazo
Planificación a largo plazo
Planificación a mediano plazo
Planificación a corto plazo
Cola de Listos para ejecutar
Monitor básico
Orden de ejecución
Políticas de Asignación
Funciones del scheduler
Planificación de trabajos
Planificación de procesos
Planificación de hilos
Tareas del planificador de trabajos
Proceso Padre
Batch
Interactivo
Multiprocesadores
Procesamiento distribuido
Planificador
Dispatcher
Procesamiento en Tiempo Real
Planificación por prioridad
Planificación de multiprocesadores
Procesamiento en paralelo
Planificación en Tiempo Real
Switcher
Procesadores homogéneos
Planificación por torneo
Cambio de contexto
Procesadores heterogéneos
Planificación preemptive
Proceso nulo o vacío
Multiprocesadores débil acoplados
Planificación non preemptive
Algoritmos de Planificación
Multiprocesadores fuertemente acoplados
Mecanismos de Planificación
Comparación de algoritmos
Granularidad
Equidad
Eficiencia
Rendimiento
Productividad
Equilibrio de recursos
Balance de carga
Planificación con múltiples colas
Políticas de Planificación
Índice de Servicio
Tiempo de respuesta
Tiempo de retorno
Tiempo de ejecución
Tiempo de servicio
Tiempo de espera
Tiempo de respuesta
Factor de ponderación
El trabajo mas corto primero
Primero en llegar Primero en servirse
Inanición
Envejecimiento
Prioridad interna y externa
Bloqueo indefinido
Evaluación Analítica
Multiservidor
Modelo de colas
Formula de Little
Modelo determinístico
Estado de un proceso
Estado Nuevo
Estado listo para ejecutar
Estado ejecutando
Estado bloqueado
Estado suspendido
Estado completado
Estado admitido
Estado Zombi
Estado de un Hilo
Estado activado
Estado en invernación
Estado dormido
Estado creado
Cola de listos

ACRÓNIMOS USADOS EN ESTE MÓDULO

S.O./ SO	Sistema Operativo
JCB	Job Control Block
PCB	Process Contol Block
TXT	Texto
M.C.	Memoria central
I/O	Input/Output
E/S	Entrada / Salida
c/u	cada uno
CPU	Central Processin Unit
MODEM	Modulación - desmodulación
J.C.L.	Job Cotrol Languge
TTY	Teletype
SYSCALL	System call
p_stat	priority- status
PSWP	priority swapper
swtch	switcher
μseg.	microsegundo
nseg.	nanosegundo

FIFO	First In First Out
FCFS	First Come First served
SJN	Shortest Job Next
SJF	Shortest Job First
RR	Round Robin
QT	Quantum de Tiempo
CTSS	Computer Time Shared System
runq	run queue
p_nice	priority nice
p_cpu	priority - cpu
p_link	priority - enlace
p_usr	priority user
p_pri	priority prior
curpri	current priority
p_wchan	priority wait change
SPN	Shortest Process Next
SRT/SRTF	Shortest Remaining Time First
HRRN	Highest Response Ratio Next
VRR	Virtual Round Robin

Anexo 3.a: Aplicaciones de políticas de planificación en distintos sistemas operativos.

a) Planificación de procesos en Linux

La parte del Sistema Operativo encargada de decidir quien será el siguiente proceso a ejecutarse es un demonio que se ejecuta en modo protegido como un componente más del núcleo de Linux.

Linux utiliza muchas estrategias de selección para asegurar el equilibrio en la selección. Estos algoritmos deben resolver varios objetivos conflictivos tales como:

- Respuesta rápida a los procesos.
- Buena transición para los trabajos en segundo instancia (background).
- Evitar la inanición de los procesos.
- Reconciliación de las necesidades de procesos de baja y alta prioridad

La selección de Linux está basado en el concepto del Tiempo Compartido. Muchos procesos son permitidos ejecutarse concurrentemente, lo que significa que el tiempo del uso del procesador está divido en partes, cada una destinada para cada proceso ejecutable. Por supuesto, un único procesador puede ejecutar solo un proceso en un momento dado. Si la ejecución de un proceso no termino cuando su tiempo o *quantum* expira, un proceso cambiador (**switcher**) toma lugar. Tiempo-compartido se basa en interrupciones por tiempo y la transparencia de procesos. Ningún código es necesario en los programas para asegurar el tiempo-compartido del Procesador.
La política de selección está también basad en un ranking de procesos de acuerdo a su prioridad.

Algoritmos complicados son usados usualmente para derivar la prioridad dada a un proceso, pero el resultado final es el mismo: cada proceso tiene asociado un valor que denota la asignación apropiada del Procesador.

En Linux, la asignación de prioridad de los procesos es dinámica. El selector mantiene un registro de lo que hacen los procesos y ajusta sus prioridades periódicamente, de este modo los procesos a los que se le ha negado el uso del Procesador por un largo intervalo de tiempo son acelerados mediante el incremento de su prioridad. También, los procesos que ejecutaron por un largo intervalo de tiempo, son penalizados a través del decrecimiento de su prioridad.

Cuando se habla de selección de procesos, estos últimos son clasificados en ligados a -E/S o ligados al Procesador. Los primeros hacen un fuerte uso de los dispositivos de E/S y pasan mucho tiempo esperando por el finalización de una operación, por el contrario, la segunda clasificación hace referencia a procesos que utilizan mucho al procesador y requieren mucho tiempo de ejecución.

Otra clasificación alternativa distingue entre 3 clases de procesos:

Procesos interactivos: Este tipo de procesos están constantemente esperando información que el usuario debe ingresar a través del teclado o por el mouse. Cuando una entrada es recibida el proceso debe levantarse rápidamente o el usuario verá al sistema colgado o sin respuesta. Típicamente, el tiempo promedio de espera debe caer entre 50 y 150 ms. La varianza de tal tiempo debe también estar destinada o el usuario encontrará errático al sistema. Los programas interactivos son los editores de texto, las aplicaciones gráficas, etc.

Procesos por Lotes: Estos no procesos no necesitan al usuario, usualmente se ejecutan en segundo plano. Desde que muchos procesos no necesitan ser muy rápidos en ejecución y finalización son penalizados por el selector. Los programas por lotes más comunes son los compiladores de lenguajes, los motores de búsqueda, etc.

Procesos de tiempo real: Estos tiene muchos requerimientos de selección. Estos procesos no deberían estar bloqueados por un proceso de menor prioridad, deben tener corta respuesta y, lo más importante,

estos tiempos de respuesta deberían tener el mínimo tiempo de varianza. Algunos ejemplos de estos programas son las aplicaciones de video y sonido, controladores robóticas, etc.

Las dos clasificaciones son independientes.
En Linux, el selector son explícitamente reconocidos pero los interactivos y por lotes no. Para ofrecer una buena respuesta a las aplicaciones interactivas, Linux implícitamente favorece a los procesos que se ligan con los dispositivos de E/S por sobre los que se atan al Procesador.

Los programadores pueden cambiar los parámetros de selección a través de llamados al sistema, que se ilustran en la siguiente tabla:

Llamadas al sistema	Descripción
nice( )	Cambia la prioridad un proceso convencional.
getpriority( )	Obtiene la máxima prioridad de un grupo convencional de procesos.
setpriority( )	Coloca la prioridad de un grupo de procesos convencionales.
sched_getscheduler( )	Obtiene la política de selección de un proceso.
sched_setscheduler( )	Coloca la política de selección y prioridades de un proceso.
sched_getparam( )	Obtiene la prioridad de selección de un proceso.
sched_setparam( )	Coloca la prioridad de un proceso.
sched_yield( )	Abandona voluntariamente al procesador sin bloquearlo.
sched_get_ priority_min( )	Obtiene el valor de mínima prioridad de una política.
sched_get_ priority_max( )	Obtiene el valor de máxima prioridad de una política.
sched_rr_get_interval( )	Obtiene el valor del tiempo *quantum* para una política Round Robin.

La mayoría de las llamadas al sistema mostradas en la tabla anterior pertenecen a los procesos de Tiempo-real, para poder crear aplicaciones. Pero, Linux no soporta las mayores demandas de las aplicaciones de Tiempo-real.

Procesos expropiativos:

Los procesos bajo Linux son no apropiativos. Esto significa que si un proceso entra en el estado de ejecución (task_running), el kernel chequea si su prioridad dinámica es más grande que la prioridad del proceso actualmente ejecutándose. En el caso de que así sea, la ejecución del proceso actual es interrumpida y el selector es invocado para seleccionar otro proceso par ejecutar (usualmente el proceso que justo acaba de entrar en listos). Por supuesto, un proceso puede también ser expropiativo cuando su tiempo de ejecución expira.

Hay que tener en cuanta que un proceso expropiativo no significa suspendido, ya que este permanece en la cola de listos, pero deja de usar el Procesador.

Algunos sistemas operativos realmente Tiempo-compartido utilizan kernel expropiativo, lo que significa que los procesos ejecutándose en modo kernel pueden ser interrumpidos luego de una instrucción, como si fuera en modo usuario. El kernel de Linux no es expropiativo, lo que significa que un proceso puede ser expropiativo solo cuando se ejecuta en modo usuario, si lo hace en modo kernel no se interrumpe ni se

expropia. El diseño de los kernels expropiativos son mucho más fácil porque la mayoría de los problemas de sincronización envueltos en las estructuras de datos del kernel son fácilmente evitados.

Duración de un Tiempo de proceso (Quantum)

La duración de un quantum es crítico para la performance de los sistemas. Este debería ser ni muy largo ni tampoco muy corto.

Si la duración de un quantum es muy corto, el overhead del sistema causado por los cambios de tarea se hacen excesivamente muy altos. Pero si la duración del quantum es demasiado larga, los procesos no parecen ser ejecutados al mismo tiempo.

La selección de la duración del quantum es siempre un compromiso. La regla adoptada por Linux es: elegir una duración tan larga como sea posible, mientras que se mantenga un buen tiempo de respuesta del sistema.

El algoritmo de Selección

La selección de Linux trabaja dividiendo al tiempo del Procesador en *epochs*. En un solo *epochs,* cada proceso tiene especificado un quantum cuya duración es computada cuando el *epochs* comienza. En general, diferentes procesos tienen diferentes quantum asignados. El valor del quantum es el máximo. El tiempo del procesador es asignado a los procesos en dichos *epochs.* Cuando un proceso ha sobrepasado su quantum, es quitado del procesador y reemplazado por otro proceso ejecutable. Por supuesto, un proceso puede ser seleccionado varias veces por el selector en el mismo *epochs*, tan pronto como su quantum no haya expirado. El *epoch* termina cuando todos los procesos ejecutables hayan acabado su quantum, en ese caso, el selector recalcula las duraciones de los quantums de todos los procesos y comienza otro *epoch.*

Cada proceso tiene una base de duración del quantum. Este es el tiempo asignado por el selector al proceso en caso de que ya haya acabado su quantum anterior en otro epoch.

El usuario pude cambiar esto usando las llamadas al sistema: nice() y setpriority() . Un nuevo proceso siempre contiene el quantum base de su propio proceso padre.

Para seleccionar el proceso a ejecutar, el selector del Linux debe considerar la prioridad de cada proceso. Hay dos tipos de prioridad:

Priorida estática: Este tipo es asignada por los usuarios a los procesos de tiempo-real y tiene rango desde 1 hasta 99. Nunca es cambiada por el selector.

Prioridad dinámica: Este tipo es usado por procesos convencionales, es esencialmente la suma de los quantum base y el número de ciclos del tiempo del Procesador que quedan al proceso antes de que el quantum expire en dicho epoch.

Por supuesto, la prioridad estática de los procesos en tiempo real es siempre mayor a los procesos de prioridad dinámica. El selector comenzará a ejecutar los procesos convencionales cuando no haya ningún proceso en tiempo-real para ejecutar.

El Selector de Linux/SMP

El selector del Linux deber ser modificado para soportar la arquitectura del multiprocesador simétrico.

Cuando el selector calcula el quantum de los procesos para ejecutar, debería considerar si ese proceso estaba previamente ejecutándose en el mismo Procesador o en otro. Un proceso que se ejecutó en el mismo Procesador es siempre preferid desde el punto que el caché por Hardware que posee el Procesador podría todavía incluir datos útiles. Esta regla ayuda a reducir el número de errores de cache.

Para poder alcanzar una buena performance, Linux/SMP adopta una regla empírica para resolver el dilema. La opción adoptada siempre es un compromiso y siempre depende principalmente del tamaño del caché integrado a cada procesador. Cuanto más largo es el caché, más conveniente es mantener enlazado el proceso con ese Procesador.

b) Planificación de procesos en Mach

Mach es un ejemplo de sistema operativo basado en un microkernel, que provee gestión de tareas/hilos (task/threads), multitarea multihilo, soporte multiprocesador y comunicación entre procesos con mensajes mediante puertos de acceso. Fue uno de los pioneros en la implementación de hilos. Mach es un sistema operativo portable a diversas arquitecturas paralelas y es compatible con UNIX. La estructura y diseño de Mach ha servido como prototipo para el desarrollo de nuevos sistemas operativos basados en microkernel, como OSF/1, Windows NT o SunOS, de ahí su enorme importancia.

Un proceso en Mach consiste en un espacio de direcciones y un conjunto de hilos ejecutándose sobre dicho espacio. Así pues, los procesos son pasivos. La ejecución está asociada a los hilos, que son dinámicos. Los procesos agrupan todos los recursos comunes a un grupo de hilos que cooperan entre sí.

La ejecución en multiprocesadores ha influido en el sistema de planificación de Mach. Lo que se hace es dividir los procesadores del sistema en conjuntos de procesadores mediante software. Cada CPU pertenece a un único conjunto. Un hilo se puede asignar a un conjunto de procesadores. De esta forma tenemos un conjunto de procesadores con un conjunto de hilos asociados para ejecución. La función del algoritmo de planificación es asignar los hilos a las CPU del conjunto de forma justa y eficaz. Cada conjunto de procesadores está aislado de cualquier otro conjunto de procesadores. Empleando este mecanismo los procesos tienen el control de sus hilos. Un proceso puede asignar un hilo de cálculos intensivos a un conjunto de procesadores dedicados, sin hilos adicionales, garantizando la ejecución continua del hilo. El proceso puede reasignar los hilos a otros conjuntos de procesadores dinámicamente, de forma que se mantenga la carga de trabajo balanceada.

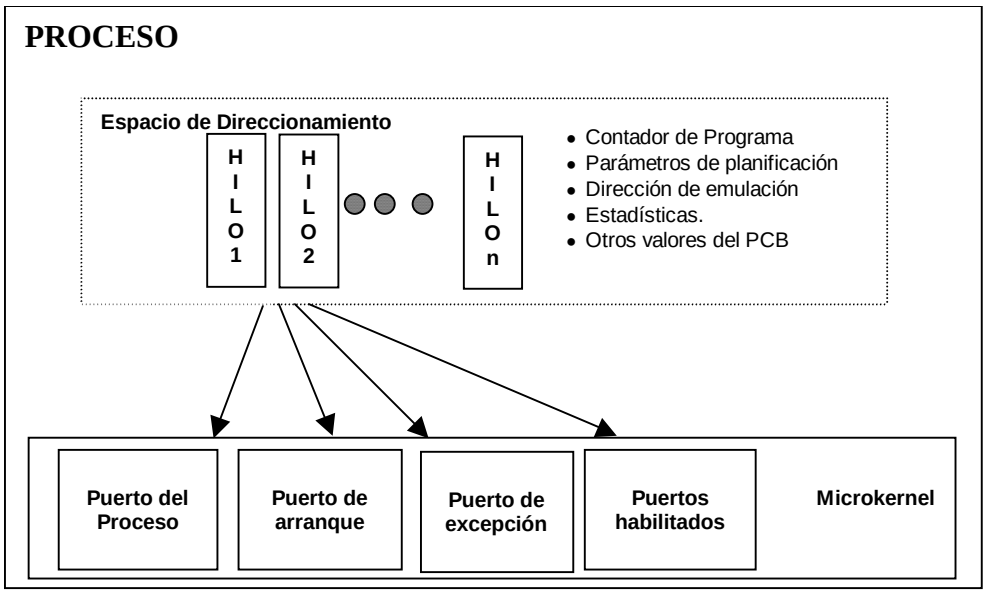


Figura 3.22. Un proceso en Mach

La planificación de hilos en Mach se basa en un sistema de prioridades inverso del 0 (máxima) al 31 (mínima) como en UNIX. Cada hilo tiene tres tipos de prioridad asignadas:

- **Prioridad base:** Es la prioridad establecida por el hilo inicialmente.
- **Prioridad máxima:** Es el mínimo valor (máxima prioridad) que el hilo puede establecer como prioridad base.
- **Prioridad actual:** Es la prioridad real que el núcleo calcula como la suma de la prioridad base y una función que depende del tiempo de utilización reciente de la CPU por el hilo.

  prioridad_actual = prioridad_base + funcion(Uso_reciente_CPU_hilo)

Cada conjunto de procesadores tiene asociado un vector de 32 colas de ejecución, correspondientes al rango de prioridades 0-31. En estas colas se encuentran sólo los hilos en estado ejecutable. Cada cola de ejecución tiene asociada tres variables:

- **Mútex:** se emplea para cerrar el acceso al vector de las colas, y garantizar que la estructura de datos es manejada por una sola CPU a la vez de todo el conjunto de procesadores.

- **Contador:** es un contador del número total de hilos en todas las colas del vector. Si su valor es 0, no existen trabajos a realizar.
- **Hint:** es un indicador de la posición del hilo con mayor prioridad.

Este vector de colas de ejecución para un conjunto de procesadores se llama *vector de colas de ejecución global*. Pero además cada CPU tiene su propio *vector de colas de ejecución local*. Estas colas de ejecución local contienen aquellos hilos que tienen su ejecución exclusiva en dicha CPU. Estos hilos no se encuentran en una cola de ejecución global.

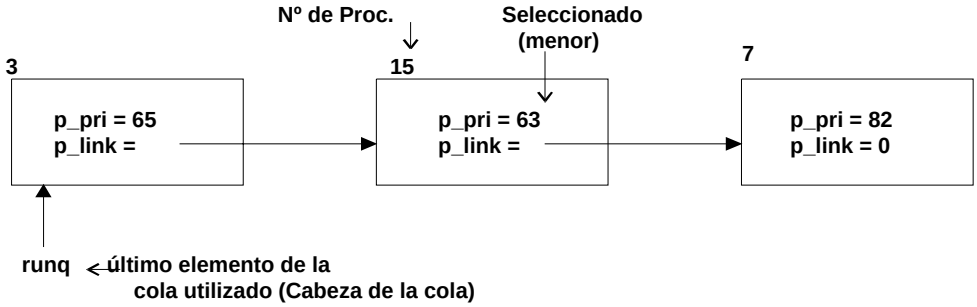
El algoritmo básico de planificación es el siguiente:

- Si un hilo se bloquea, finaliza o agota su quantum de tiempo, la CPU donde se ejecutaba busca primero en su vector de colas de ejecución local, inspeccionando la variable contador, para ver si existen hilos activos. Si el contador es distinto de 0 comienza la búsqueda a partir de la cola de prioridad indicada por la variable hint, por ser de máxima prioridad.
- Si no existen hilos en las colas locales, se repite el algoritmo para el vector de colas de ejecución glocal, pero si tampoco existen hilos, se ejecuta un hilo especial inactivo hasta que exista un hilo listo.
- Si se localiza un hilo ejecutable, se planifica y se ejecuta durante un quantum. Al terminar el quantum se comprueban las colas locales y glocales para ver si existen otros hilos ejecutables de su misma o mayor prioridad, teniendo en cuenta que todos los hilos de la cola local tienen mayor prioridad que los hilos de las colas glocal. Si se localiza un candidato se produce una conmutación de hilos, pero si no es así, se ejecuta de nuevo el hijo durante otro quantum de tiempo.
- El quantum en los sistemas multiprocesador puede ser variable. Cuanto mayor es el número de hilos y menor el número de CPU, menor será el quantum asignado. Esto proporciona tiempos de respuesta bajos para solicitudes cortas, incluso con una gran carga de trabajo, y proporciona una alta eficacia en casos de poca carga.
- Con cada pulso de reloj, la CPU incrementa el valor de la prioridad del hilo en ejecución, lo que hace bajar su prioridad. Los contadores de prioridad de los hilos en espera se decrementan en su valor cada cierto tiempo, lo que hace subir su prioridad para su planificación.

Además de las prioridades, Mach proporciona otra forma de control de planificación adecuada para un conjunto de hilos que cooperan en un trabajo, el *gancho (hook)*. Esto le proporciona a los hilos un control adicional sobre la planificación. El gancho es una llamada al sistema que permite a un hilo reducir su prioridad al mínimo durante unos segundos, lo que da a otros hilos la oportunidad de ejecutarse. Transcurrido ese tiempo, el hilo recupera su prioridad anterior. Además esta llamada permite especificar el hilo sucesor si se desea. Por ejemplo, un hilo puede enviar un mensaje a otro, entregar la CPU y solicitar que el hilo receptor del mensaje se pueda ejecutar a continuación. A este mecanismo se le suele llamar *planificación manos-fuera*, ya que se salta las colas de ejecución. Empleado de forma adecuada puede mejorar el rendimiento de una aplicación. Es empleado en ocasiones por el núcleo para optimizar algunas acciones.

c) Ejemplo de planificación utilizada en el S.O. UNIX.

Utilizaremos el siguiente esquema para representar la simulación de los procesos que compiten por CPU:



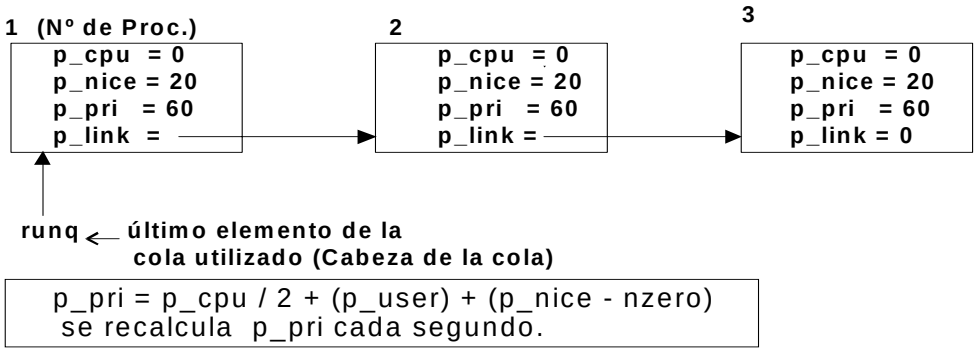
p_pri = es la prioridad del proceso. Su rango está entre $0 \leq$ p_pri ≤ 119.
Se utiliza para seleccionar el proceso que tiene la menor p_pri.

En UNIX la prioridad mas alta le corresponde el número mas bajo almacenado en p_pri

p_cpu = Computated time / Elapsed time
- Factor primario en la prioridad del proceso
- incrementa cada clock TICK para el proceso en ejecución, NO EXCEDIENDO UN VALOR MÁXIMO DE 80
- dividido por 2 cada segundo parra todos los procesos, incluso el que está en uso de la CPU.

p_nice = Clase de prioridad determinada por el tipo de usuario.
Rango entre $0 \leq$ p_nice ≤ 39 (Default 20) (0-19 Super User)

Ejemplo 1
Asumimos:
- TRES PROCESOS
- TODOS EJECUTABLES EN ESE INSTANTE DE TIEMPO
- RUN FOR EVER
- 60 CLOCK TICKS PER SECOND



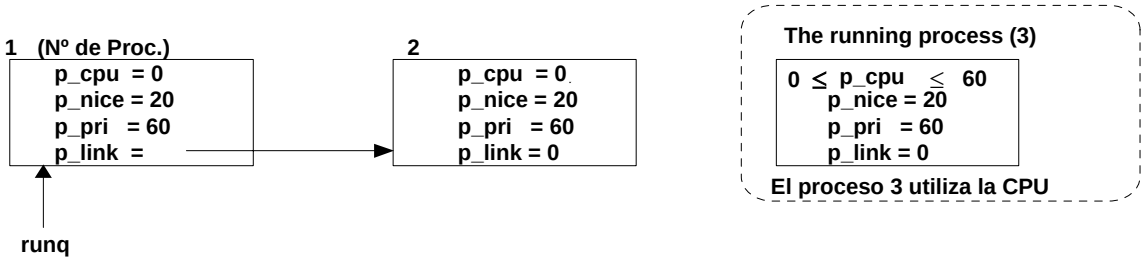
p_pri = p_cpu / 2 + (p_user) + (p_nice - nzero)
se recalcula p_pri cada segundo.

Se usa un reloj por hardware llamado **clock tick** que equivale un sesenta avos de segundo
Cada clock tick solo incrementa p_cpu del proceso en ejecución no excediendo de un valor máximo de 80.

Cada segundo se ejecuta el siguiente código para todos los procesos:

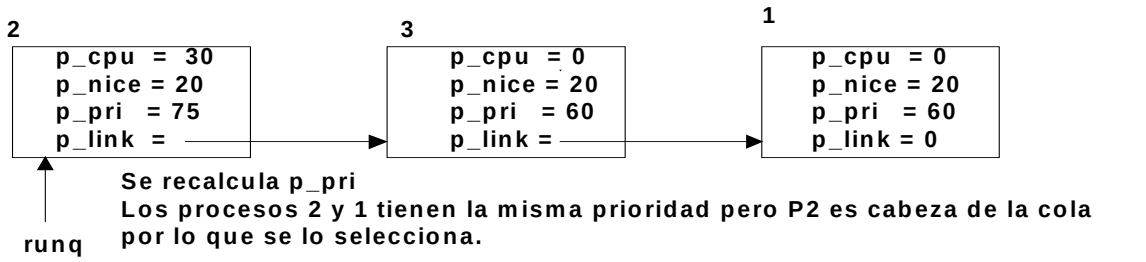
```
p_cpu = p_cpu /2
Si  p_pri ≥ (p_user - nzero)
        p_pri = (p_cpu / 2) + (puser) + (p_nice - nzero)
Arreglar para activación del switcher
```

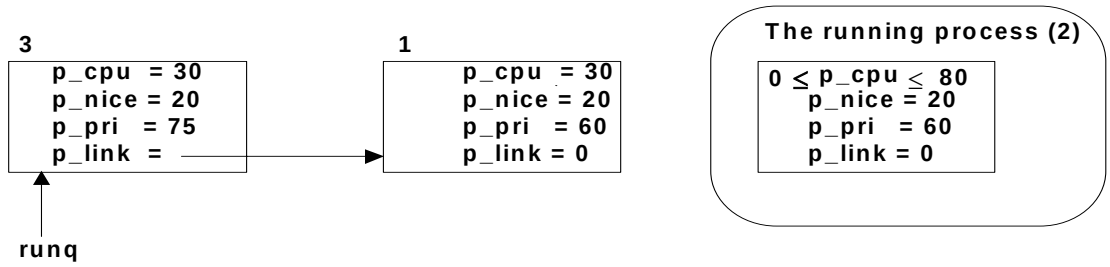
Hay 2 grupos (puser - nzero)
- entre 0 y 39 privilegio de los procesos del Sistema Operativo (System privilege)
- 40 y 119 privilegio de los usuarios y sus procesos.
- Default: puser == 60
  nzero == 20



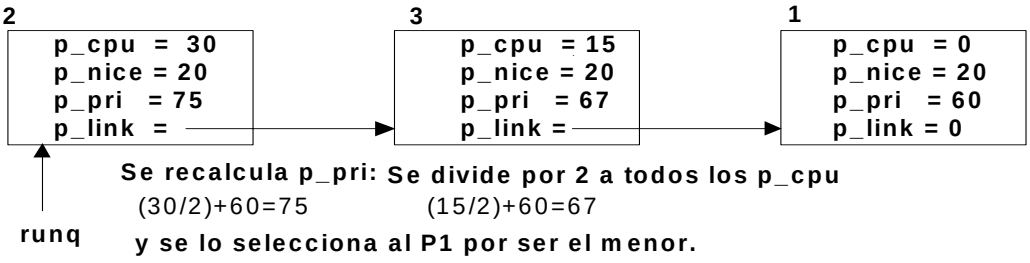
Todos los procesos tienen igual prioridad. Como el proceso 3 está ubicado como cabeza de la cola, entonces es seleccionado.
p_cpu (del 3) se incrementa cada clock tick.

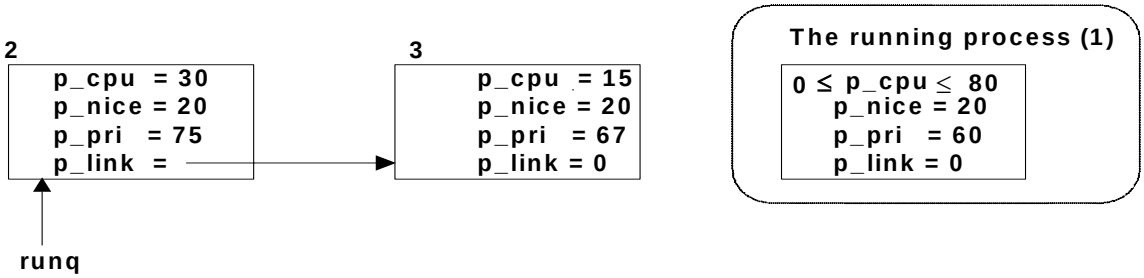
- DESPUES DE 1 SEGUNDO:





- DESPUÉS DE 2 SEGUNDOS:





En este ejemplo observamos que se implementa el algoritmo Round Robin, dado que los tres procesos se van a alternar en el mismo orden.

Ejemplo 2

En este ejemplo, los procesos se diferencian en que los usuarios tienen prioridades distintas, por lo que se modifican las veces que acceden a usar la CPU.

El Algoritmo esta basado en las prioridades.

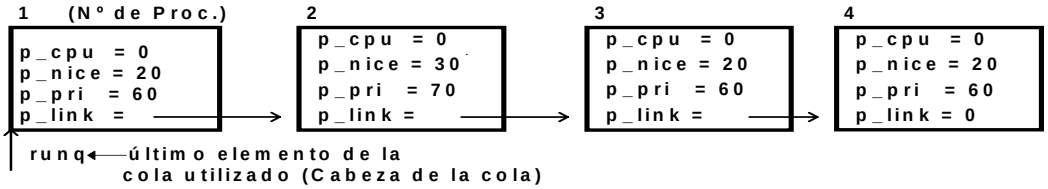
Se tiene dos procesos con p_nice = a 20, otro con 30 y el último con p_nice = a 0 (Super-Usuario, o sea el que administra el Sistema).

Observamos que las prioridades (p_pri) se calculan mediante la fórmula:

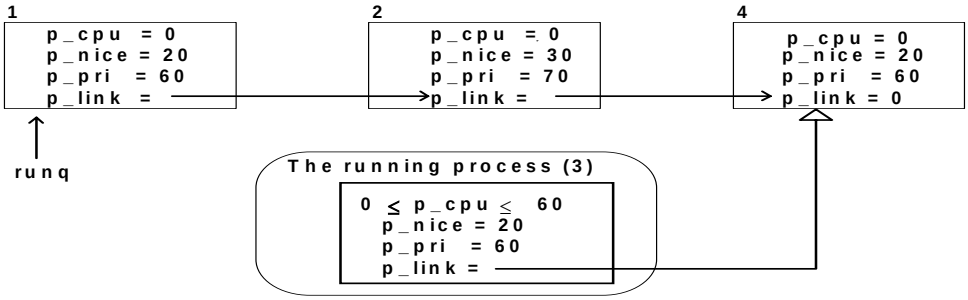
p_pri = p_cpu/2 + puser + (p_nice - nzero)

Asumimos:
- CUATRO PROCESOS
- TODOS EJECUTABLES EN ESE INSTANTE DE TIEMPO
- RUN FOR EVER
- 60 CLOCK TICKS POR SEGUNDO

El proceso 3 es seleccionado porque tiene la mayor prioridad. Después que fue hecha la selección el proceso 3 es removido de la cola de ejecución y es colocado en uso de la CPU.
Obsérvese que el valor de p_cpu del proceso 3 se incrementará con cada clock tick en uno.



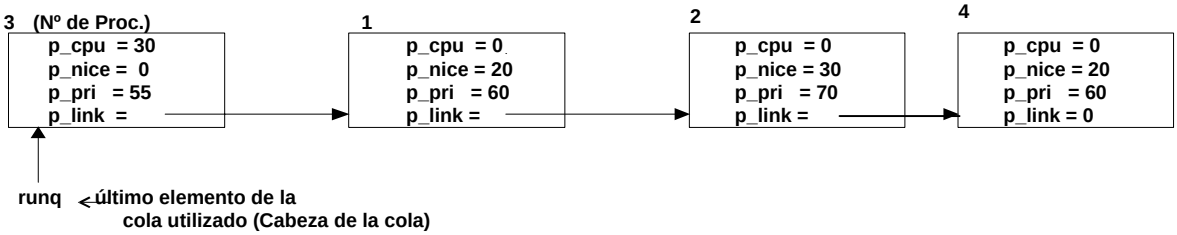
p_pri = p_cpu/2 + puser + (p_nice - nzero) =



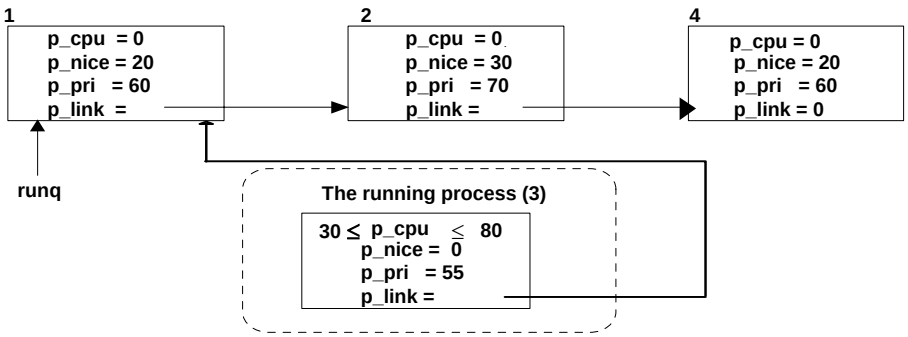
Se selecciona 3. Tiene menor p_pri, o sea mayor prioridad.

- DESPUÉS DE 1 SEGUNDO:

Las prioridades de todos los procesos son recalculadas. Obsérvese que a todos los procesos (incluido el 3) se divide por 2 antes de que se recalculan las prioridades.
El proceso en ejecución es sacado de la CPU (switch out) a pesar de que inmediatamente se lo vuelve a seleccionar por ser el de menor prioridad (p_pri=55).



p_pri = p_cpu/2 + puser + (p_nice - nzero)= (30/2)+ 60-20= 55

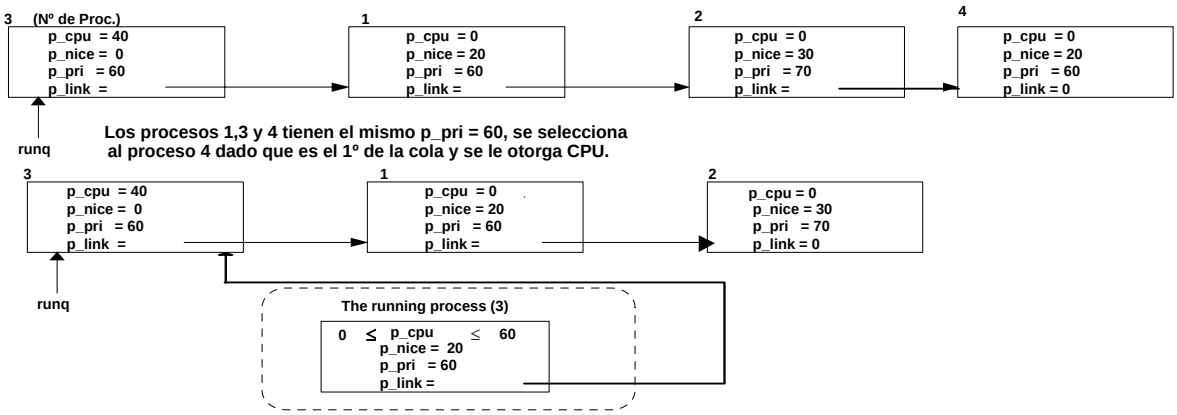


Se selecciona 3. Tiene menor p_pri, o sea mayor prioridad.

Obsérvese que p_cpu del proceso entra con 30 y no puede exceder de 80
por lo que cuando termina el segundo de uso de CPU quedará en 80.

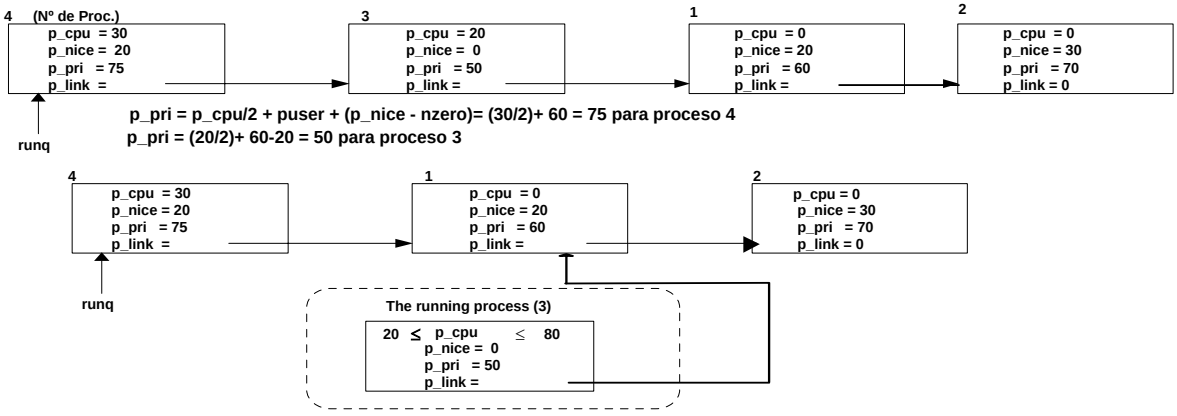
- DESPUÉS DE 2 SEGUNDOS:

El proceso 3 es sacado de la CPU y colocado en la Cola de listos. Esto ocurrió por haberse producido una interrupción por reloj (fin del Quantum de tiempo).



- DESPUÉS DE 3 SEGUNDOS:

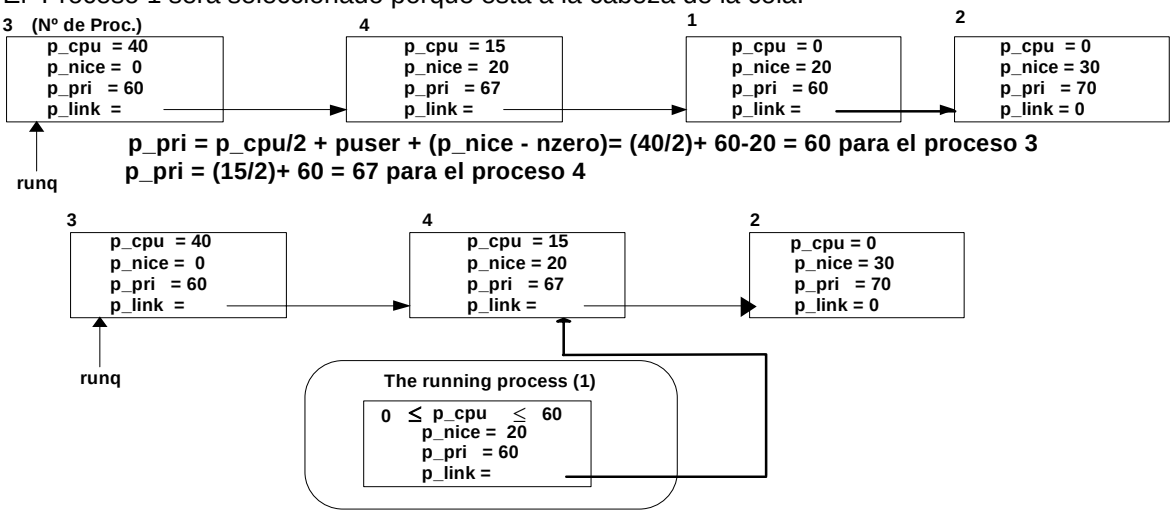
Se produce el recalculo de p_pri que hace al proceso 3 ser el próximo candidato a usar CPU.



Observación: el Proceso 3 nunca excede su p_pri de 60.

- DESPUÉS DE 4 SEGUNDOS:

El Proceso 1 será seleccionado porque está a la cabeza de la cola.



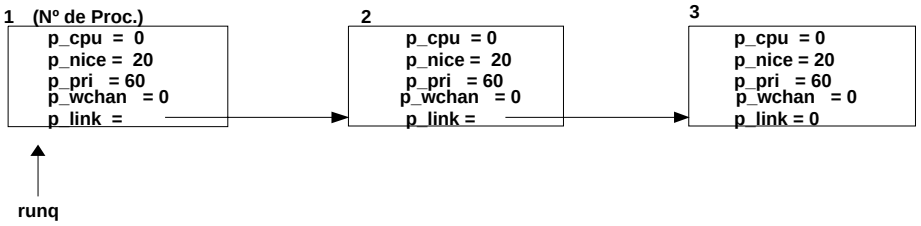
Nótese que el proceso 2 no ejecuta debido a la particularidad de la distribución del valor de p_nice. En realidad, esto no ocurre, porque el proceso 3 eventualmente debe terminar en algún momento.

Ejemplo 3 con I/O y CPU Bound Process

UNIX utiliza CURPRI, una variable del sistema que representa la prioridad del proceso en ejecución corriente o sea que está en uso de la CPU.

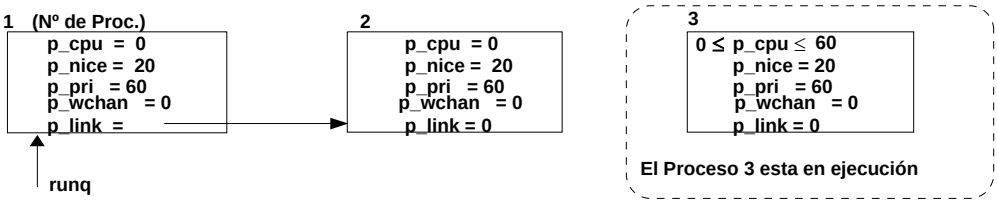
Supongamos que todos los procesos tienen la misma prioridad (p_pri = 60). Se selecciona el proceso 3 porque esta al comienzo de la cola y **curpri** es colocado en 60.

- Proceso 1 : Terminal I/O
- Proceso 2 : Disk I/O
- Proceso 3 : CPU Bound
- 60 clock ticks por segundo



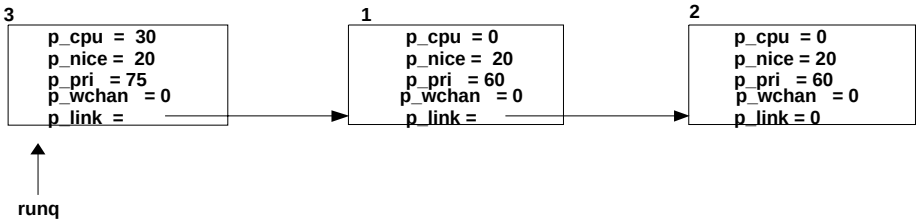
- curpri = 60 Con la Interrupción de reloj de 1 segundo el proceso en ejecución (current process) sale.

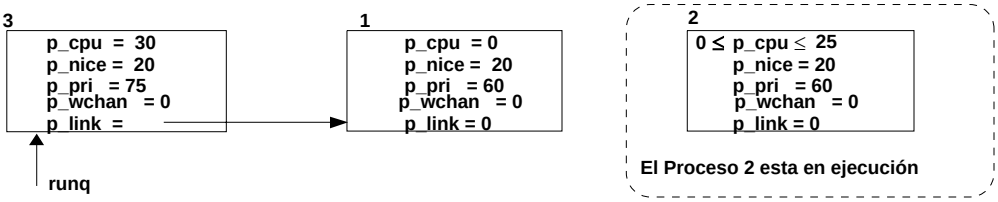
Notar que p_pri y p_cpu han cambiado para el proceso 3.



- DESPUÉS DE 1 SEGUNDO:

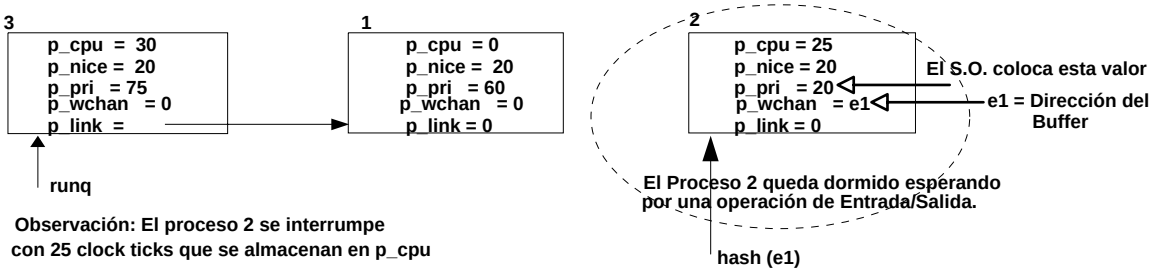
Al cabo del primer segundo se modifican los valores de p_pri y p_cpu del proceso 3, luego se selecciona el proceso 2 para ejecutar y curpri es seteado en 60.



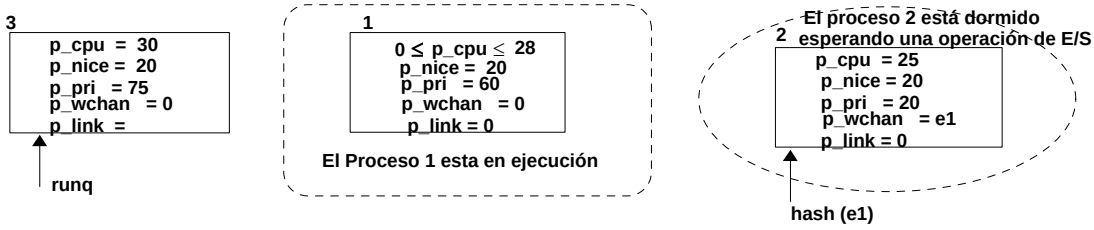


Proceso 2 se va a Dormir a los 25 clock ticks por pedir una operación de E/S sobre disco (Disk I/O request), como consecuencia su prioridad es cambiada a 20 (Block I/O) por el sistema.

El identificador de evento (buffer address) es registrado en p_wchan y el proceso es colocado en la cola adecuada y la operación de I/O es ejecutada por el procesador de I/O asincrónico.

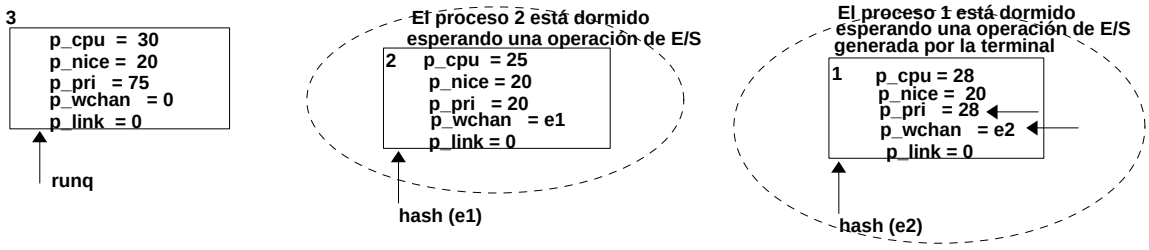


El S.O. selecciona al próximo proceso candidato para ejecutar de la cola.

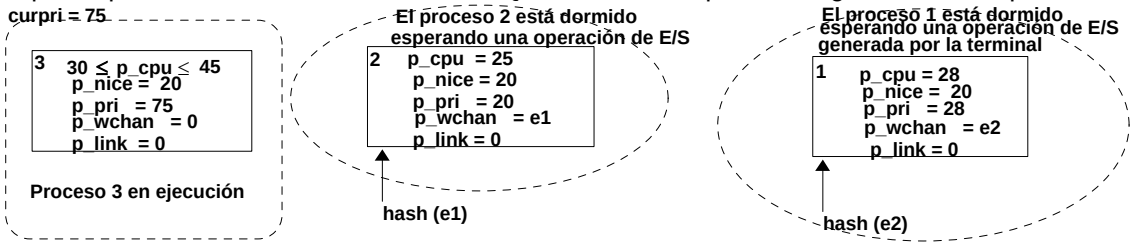


Proceso 1 es seleccionado. La cola de ejecución contiene sólo el proceso 3.

Después de 28 ticks, requiere una operación de E/S (Terminal input request) por lo que también se bloquea (duerme) esperando hasta que se complete dicha operación generada por la terminal. Como resultado, su prioridad es cambiada a 28 por el S.O. Este valor corresponde a la prioridad de entrada de terminal (terminal input priority),. El identificador de evento (dirección de la terminal o terminal address e2) es colocado en p_wchan y el proceso es introducido en la apropiada cola de Dormidos. El pedido de entrada es realizado por el TTY mediante un procesador asincrónico.

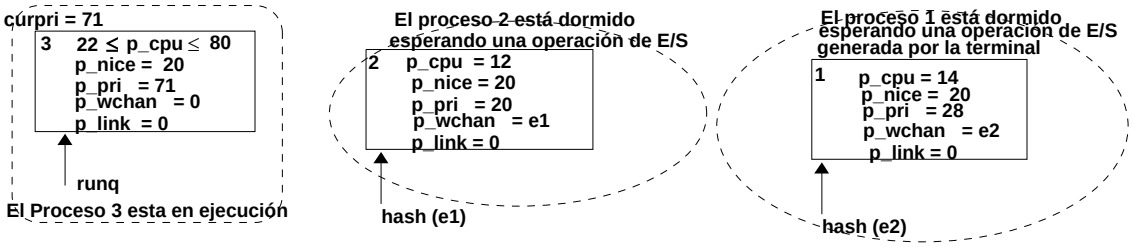


El único proceso ejecutable es el 3. Se lo selecciona y su curpri = 75 que es la prioridad del proceso 3 . Notar que dispone de 15 ticks remanentes del Quantum de tiempo de 1 segundo interrumpido.

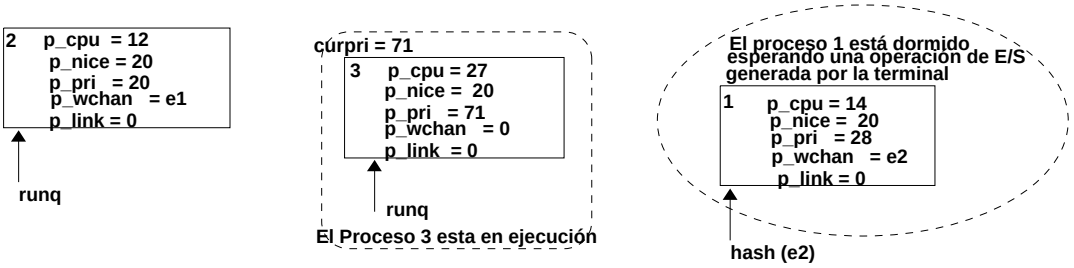


- DESPUÉS DE 2 SEGUNDOS:

Se divide p_cpu por 2 (todos los procesos incluidos los dormidos).
Proceso 3 usa 15 clock ticks y por eso la p_cpu tiene 22 (45/2).
La prioridad del proceso 1 es recalculada . Las prioridades de los procesos 2 y 3 no son recalculadas, debido a que tienen System Priority (< 40)
El Proceso 3 es switcheado (sacado), ocurrido por la interrupción de clock (1 segundo). Esto no ocurre porque el proceso 3 es el único proceso ejecutable entonces sigue en uso de la CPU. Se le asigna curpri = 71.

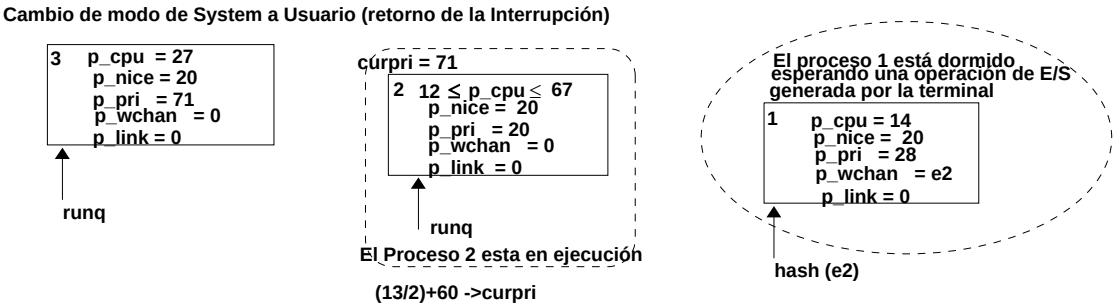


Llega la interrupción de I/O del Disco después de 5 ticks. Proceso 3 ejecuta la rutina de interrupción (en modo system). Como consecuencia del tratamiento de la interrupción se ejecuta **Despertar (wakeup)** que es un Syscall. Entonces el proceso 2 es sacado de la cola de dormidos y colocado en la cola de ejecución. p_wchan del proceso 2 es limpiado. El despertar del proceso 2 lo ha encontrado con prioridad = 20 que es más alta que el proceso en uso de la CPU (71), entonces ajusta la bandera (flag) **runrun.**



Al retornar del Interrupt handler, la bandera runrun es testeada y se encuentra que está seteado. El proceso 3 en ejecución es colocado en la cola de ejecución y sacado de CPU y se selecciona el proceso 2 para ejecutar. Se prefija a Curpri = 20.
El completar el SYSCALL resulta que el proceso 2 reasume su ejecución en modo user. La prioridad del proceso 2 es recalculada y curpri es cambiado.

- Cambio de modo sistema a modo user (en el retorno de la interrupción)
- curpri = 20



Secuencias de Despertar

- Se ha completado más I/O (Asincrónicamente)

- Runtime de interrupción cuando se ha completado la I/O
- Despertar
- Switch in

- Demora por tiempo desconocido entonces Despertar y switch.

El despertar representa la adición del proceso o procesos a la cola de listos. Para después ser activado por la rutina de switch, entonces el proceso es seleccionado y reasume su ejecución.

Razones para dormir:

- Principalmente como resultado de SYSCALLS espera por:

- completado de I/O.
- Recursos compartibles escasos (Ej.: canal de I/O).

- Obtención de recursos no compartibles (Ej.: un i nodo lockeado hasta que sea liberado).
- Eventos de sincronización (Ej.: espera del padre por la terminación de hijo(s).

Eventos

- Identificado por un p_wchan
- Cada evento tiene asociada una cola de Dormidos
- Más de un proceso puede estar Dormido en la misma cola esperando por eventos.
- Identificado por un p_wchan

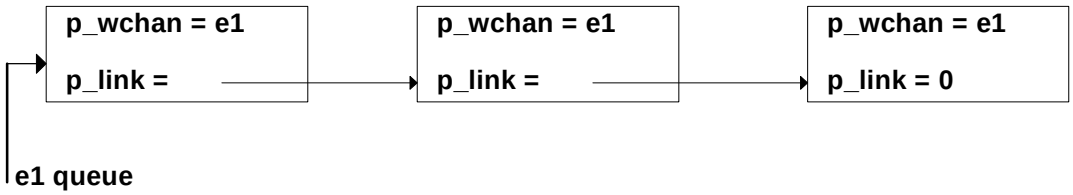
- p_wchan es el campo identificador de evento pero no es el evento.
- El único requerimiento de un identificador de evento es que debe ser único. Por ejemplo el identificador de un evento de entrada de terminal es diferente de un identificador para lectura de disco.
- Direcciones de estructuras de datos son usadas para designar identificadores de eventos, por ejemplo, un proceso en espera por entrada de terminal especifica la dirección de las estructura de TTY asociada con la terminal que tiene un identificador de evento.
- El identificador de evento es determinado por el S.O.

- Cada evento tiene asociada una cola de Dormido

- Normalmente un proceso forma parte de una cola de ejecución o de una cola de Dormidos. El campo **p_link** de un PCB o Tabla de proceso es usado para mantener las colas. El S.O. mantiene una tabla con las entradas que son cabeza de las colas (punteros a los PCB). Cada entrada es asociada con uno o más eventos.
- Entradas apropiadas para un identificador de evento se obtienen mediante hashing. Hashing es simplemente una función que acepta un identificador de evento y retorna un valor entero. Este valor es usado como un índice a la tabla de hash. Dos identificadores de evento no pueden tener la misma entrada de hash (colisión). Esto es, una cola de Dormidos puede contener procesos perteneciendo a dos o más eventos diferentes. De cualquier modo, el identificador de evento actual es almacenado en p_wchan.

- Más de un proceso puede estar Dormido en la misma cola esperando por eventos.
  - Por ejemplo, dos procesos pueden estar esperando por entrada de una misma terminal

Sleep Queues (Colas de Dormidos)



La línea identificada por e1 queue es normalmente una entrada en la hash Table. Esta lista enlazada puede potencialmente contener procesos con un p_wchan que es el mismo que otro identificador de evento (por ejemplo e2). Si esto aconteciese la búsqueda (hash) de e1 y e2 resulta en la misma entrada.

Estado de un proceso

- Registrado en p_stat.
- SRUN - Ejecutando o ejecutable.

Todos los procesos con p_stat == SRUN, excepto el proceso ejecutando, es parte de la cola de ejecución.

- SSLEEP - Durmiendo

Todos los procesos con p_stat == SSLEEP debe ser parte de la misma cola de Dormidos.
Hay otros valores que puede asumir p_stat, pero la mayoría del tiempo los procesos lo pasan en los estados SSLEEP o SRUN.

Prioridad del sistema

- Los valores están dentro del Rango de 0 a 39
  La prioridad más baja del sistema (39 para pausa) es mayor que la mayor prioridad del usuario (40 para p_nice = 0 ; p_cpu = 0)
- Solo como resultado de un Dormir.
  -Cuando un proceso se duerme su prioridad cambia. El sistema, dependiendo del evento que llevó al proceso a Dormir, impone una prioridad al proceso.
- Diferentes eventos pueden tener la misma prioridad
  -Los eventos tienen la misma categoría. Por ejemplo: dos procesos esperan por entrada de terminal (diferentes terminales) y tienen prioridad = 28.
- Asegura que los procesos que utilizan recursos críticos del sistema se ejecutan antes que otros procesos.
  - Dispone que el switcher seleccione los procesos de la cola de ejecución que tienen prioridad más alta.
  - Cuando un proceso dormido está usando recursos del sistema (Ejemplo una Entrada/Salida de disco usa un buffer del sistema) es seleccionado para que abandone el uso del recurso lo más rápido posible.
- Genera el cambio de prioridad a prioridad de user antes de retornar del SYSCALL.
  - Los procesos en no modo usuario tienen prioridad del sistema. Después que el SYSCALL se ha completado la prioridad es recalculada mediante la fórmula.
- Las prioridades del sistema no se recalculan
  - Las prioridades del sistema están protegidas del recálculo hecho en cada segundo (Dispuesto en clock, si p_pri >= PUSER - NZERO es 40, entonces p_pri es recalculado) mientras que p_cpu es incrementado cada tick.
  - La cantidad de tiempo que el proceso ejecuta con prioridad del sistema es relativamente muy corta, generalmente la última etapa del SYSCALL (Ej.: transferencia de información desde el espacio de direccionamiento del sistema al espacio de direccionamiento del usuario)

Valores de prioridad del Sistema

PSWP	0	Swapper
PINODE	10	Inodo asegurado (Locked inode)
PBIO	20	E/S Bloqueada (Blocked I/O)
PZERO	25	Lowest High Priority (Prioridad más baja)

High ↑

Low ↓

PPIPE	26	Pipe
TTIPRI	28	Entrada terminal
TTOPRI	29	Salida terminal
PWAIT	30	Espera por la muerte de hijo(s)
PSLEEP	39	Pausa
PUSER	60	Nivel de entrada del user
PIDLE	127	Idle CPU

- referencia /usr/src/uts/machine/sys/param.h

PZERO representa la unión entre alta y baja prioridad del sistema.
Los procesos cuyo valor de prioridad es igual o menor que PZERO son considerados "importantes". Por ejemplo si se duerme un proceso con prioridad menor o igual a PZERO no se lo puede ser perturbado (disturb) por una señal.

SUBRUTINA DORMIR

sleep(event, priority)

p_stat = SSLEEP
p_wchan = event
p_pri = priority
colocar al proceso en la cola apropiada de Dormir
swtch()

La subrutina dormir cambia el estado del proceso a SSLEEP, carga el identificador con el evento, cambia la prioridad del proceso a prioridad del sistema, coloca en la cola de Dormir apropiada y llama la rutina de switch. Si el proceso es introducido, su ejecución es reasumida en Dormir (después que llamó a switch()).

SUBRUTINA DESPERTAR

wakeup(event)
para todos los procesos de la cola asociados con el evento
p_wchan = 0
p_stat = SRUN
Removerlo de su cola de Dormidos y colocarlo en la cola de Ejecución
If curpri > p_pri
set runrun
request to run a swtch()

- referencia: /usr/src/uts/machine/OS/slp.c
- La subrutina despertar acepta un identificador de evento como parámetro y coloca a todos los procesos dormidos por tal evento en la cola de ejecución. Limpia el campo identificador de evento y potencialmente coloca el flag de runrun
- La llamada de Despertar no todos los procesos van a ejecutar inmediatamente.
- Otra rutina llamada Setrun, que esta en slp.c, realiza una acción similar. La diferencia entre ambas es que despertar acepta un identificador de evento y potencialmente despierta varios procesos, mientras que Setrun acepta la dirección de una entrada a la tabla de procesos (PCB) (Ej &proc[0]) y despierta solamente a un proceso.

CURPRI (current priority)

- Variable del sistema
- Es la prioridad del proceso en ejecución corriente
- Colocado en swtch() a la prioridad del proceso seleccionado
- Colocado a prioridad del user en el proceso antes que retorna de un SYSCALL

NOTAR que Curpri no es colocada en la entrada del reloj cada segundo porque sino cuando ocurre un switch entonces es seteado in swtch().

Runrun:

- Es una bandera del Sistema Operativo (System Flag).
- Cuando es prefijado, el switcher será activado en el próximo periodo cuando se cambia el modo de Sistema a modo usuario. Un cambio de contexto (switch), que no ocurre cuando el procesador está en modo sistema, elimina un conjunto de problemas que resultan del tratamiento de códigos críticos.
- Se prefija en: despertar (wakeup), setrun, clock.

- En wakeup y setrun, cuando un proceso con prioridad mas alta que el proceso en ejecución corriente o actual es despertado (p_pri < curpri).
- en clock, cada segundo.

- Reset en **swtch().**

```
23   #define   NHSQUE   64
24   #define sqhash(X) (&hsque [((int)X>>3) & (NSSQUE-1)])

25    struct proc *hsque [NHSQUE)];
26    struct proc *curproc, *runq;
```

```
 40    #define    TZERO        10

 41    sleep(chan, disp)
 42    caddr_t chan;
 43    {
 44      register struct proc *rp = u.u_procp;
 45      register struct proc **q = sqhash(chan);

 54      rp ->p_stat = SSLEEP;
 55      rp ->p_ wchan = chan;
 56      rp ->p_link = *q;
 57      *q = rp;
 73      swtch();
 94     }

 99    wakeup(chan)
100     register caddr_t chan;
101     {
103       register struct proc **q;
108       for (q = sqhash(chan); p = *q; )
109       if (p->p_wchan==chan && p->p_stat==SSLEEP) {
110           p->p_stat = SRUN;
111           p->p_wchan = 0;
113           *q = p_link
114           p->p_link = runq;
115           runq = p;
121           if (p->p_pri < curpri)
122           runrun++;
123           } else
124           q = &p->p_link;
130      }
```

Comentarios sobre las líneas:

24 a 27- definición de la tabla de hash, la función de hash, y la cola de ejecución.

44- La entrada de la tabla de proceso (PCB) del proceso que está en uso de la CPU es registrado en **rp**.

45- coloca el identificador de evento a ser utilizado cuando el proceso es colocado en la cola de dormidos.

54- registra el estado del proceso mientras duerme.

55- registra el identificador de eventos

56-57- lo coloca en la adecuada cola de dormidos.

73- conmuta a un nuevo proceso.

108- Localiza la cola de dormidos asociado con el evento.

109- Examina el campo p_wchan de todos los miembros de la cola de dormidos. Notar que procesos con diferentes p_wchan pueden tener la misma cola de dormidos como resultante de la colisión en hashing.

111- registra el estado del proceso como ejecutable.

113- Toma este proceso de la cola de dormidos.

114-115- Coloca al proceso en la cola de ejecución.

121-122- Si el proceso tiene una prioridad mayor que el proceso actual en ejecución, prefija la bandera runrun , para que cuando el switcher es activado mas tarde, el procesador transfiere de modo sistema a modo usuario.

d) Nuevos Algoritmos de Planificación de Tiempo Real

Introducción

Design-to-Time y Design-to-Criteria son algoritmos de planificación en tiempo real que están pensados para problemas en los cuales existen múltiples caminos (alternativas) para obtener una solución a un problema dado. Ambos algoritmos están pensados para trabajar con un tiempo limitado, buscando una solución suficientemente buena, no necesariamente óptima, dentro del tiempo disponible.

Design-to-Time difiere de los algoritmos "tradicionales" de planificación en el hecho de que está pensado para problemas en los que puede haber múltiples métodos para hallar una solución. Esto implica que el planificador tiene una responsabilidad extra: la generación de *alternativas*. Mientras que la mayoría de los planificadores se limitan a determinar en qué orden se deben ejecutar las tareas, Design-to-Time debe además determinar qué ejecutar.

El hecho de que Design-to-Criteria permita que se le especifique el criterio a seguir para evaluar las planificaciones hace que sea más aplicable en sistemas de Inteligencia Artificial. Esto permitiría que, por ejemplo, de acuerdo a la situación, el agente le indique al planificador cuales son los criterios que considera que es conveniente seguir.

Design-to-Time

Visión general

Es una metodología utilizada para generar planificaciones que permitan resolver problemas con restricciones de tiempo real. Existen distintos algoritmos que utilizan de una forma u otra esta metodología, adaptándola según la naturaleza del problema.

Las restricciones que se le pueden imponer pueden ser fuertes (*hard*) o débiles (*soft*).

Definiciones

Antes de sumergirnos en mayor detalle en Design-to-Time, definiremos algunos términos que son relevantes a la naturaleza de la metodología.

- ***Método de solución*:** Es una tarea o sub-tarea que puede ejecutarse para solucionar un problema dado.
- ***Método ejecutable*:** Son tareas ejecutables individuales (que no pueden descomponerse en métodos o sub-tareas).
- ***Duración*:** Es el lapso que un método ocupa para completar su ejecución.
- ***Quality*:** Es un valor que indica que tanto aportaría a la solución del problema la ejecución de cierto método.
- ***Plazo de vencimiento (deadline)*:** Son restricciones que ponen un límite al tiempo de finalización de la ejecución. En Design-to-Time los plazos de vencimiento se aplican sólo a los grupos de tareas.
- ***Aproximación*:** Se dice que un método es aproximación de otro cuando sirve para resolver el mismo problema. Generalmente las aproximaciones son métodos que generan un quality menor pero son más rápidos, por lo que son ejecutados cuando el tiempo no alcanza para ejecutar el método original.
- ***Alternativa*:** Conjunto de métodos sin ordenar.

Requerimientos

Para que se pueda usar una metodología Design-to-Time, el problema a resolver debe cumplir con los siguientes requerimientos:

- Debe haber múltiples soluciones (*métodos de solución*) para lograr un mismo objetivo.

- Debe buscarse una solución aceptable, no necesariamente óptima, ya que Design-to-Time no puede garantizar que encontrará la mejor solución.
- Los plazos de vencimiento (*deadlines*) deben ser razonablemente predecibles (no así la probabilidad de arribo de una nueva tarea).
- Se debe poder establecer como influye la ejecución de un método en la ejecución de otros.

A su vez, si el sistema debe tolerar incertidumbre (como en la mayoría de las aplicaciones del mundo real), deben cumplirse ciertas condiciones extras:

- Se debe poder monitorear la ejecución para poder detener el método y ejecutar una aproximación.
- Siempre debe haber un método de emergencia que, de ejecutarse, genere una solución mínima aceptable en poco tiempo.

Características

Si el problema reúne las condiciones enumeradas con anterioridad, Design-to-Time puede presentar una solución con la siguientes características:

- Puede respetar restricciones fuertes (hard) y / o débiles (soft).
- Está disponible en el plazo que se le dé al planificador. A medida que se le da más tiempo, el planificador puede mejorar la solución.
- Cada grupo de tareas tendrá un quality mayor que cero.
- No se ejecutará ningún método después del plazo de vencimiento (deadline) de su grupo de tareas.

Design-to-Criteria

Visión general

Design-to-Criteria es una evolución de Design-to-Time. Difiere, en primer lugar, en la forma en que evalúa las distintas planificaciones. Para realizar dicha evaluación, Design-to-Time se basa solamente en el *quality* y en la *duración*. Design-to-Criteria añade el factor *costo*. Pero la diferencia más importante es que a Design-to-Criteria se le puede especificar que criterio seguir, como por ejemplo, “darle más importancia a los plazos del grupo A que al quality del grupo B”. El planificador podría incluso negociar nuevamente este criterio con el cliente, ya se por cuestiones inherentes a uno o a otro.

Otra diferencia es que Design-to-Criteria está pensado para tratar con restricciones de tiempo real débiles (*soft*).

Design-to-Time difiere de los algoritmos “tradicionales” de planificación en el hecho de que está pensado para problemas en los que puede haber múltiples métodos para hallar una solución. Esto implica que el planificador tiene una responsabilidad extra: la generación de *alternativas*. Mientras que la mayoría de los planificadores se limitan a determinar en qué orden se deben ejecutar las tareas, Design-to-Time debe además determinar qué ejecutar.

El hecho de que Design-to-Criteria permita que se le especifique el criterio a seguir para evaluar las planificaciones hace que sea más aplicable en sistemas de Inteligencia Artificial. Esto permitiría que, por ejemplo, de acuerdo a la situación, el agente le indique al planificador cuales son los criterios que considera que es conveniente seguir.

Definiciones

A las definiciones de Design-to-Time se la agregan algunas que forman parte de la jerga de Design-to-Criteria:

- **NLE (non-local effects):** Así se denomina aquí a las interacciones entre tareas (como por ejemplo, las relaciones de habilitación, facilitación e impedimento que se definieron con anterioridad).

- **Slider:** Hace referencia a un parámetro que se puede ajustar. Slider no tiene traducción exacta al castellano, se denomina de esta forma a las barras de desplazamiento para ajustar una variable.
- **CTER (critical task execution region**): Región de ejecución de tareas crítica. Es un método que, de fallar, puede degradar el rendimiento de la planificación en su conjunto.

Especificación de los criterios

El cliente puede especificar la forma de evaluación deseada mediante sliders. Mediante estos parámetros y los valores esperados de la tarea (de quality, costo y duración), el planificador obtiene un número (rating) que sirve para indicar la preferencia de una alternativa sobre otra.

Resulta conveniente mencionar que estos sliders no permiten especificar ningún tipo de restricción fuerte (hard constraint). Design-to-Criteria no está pensado para restricciones fuertes.

Proceso intermitente

Las mejoras para proceso intermitente apuntan a aprovechar retardos (delays) que se puede dar en la ejecución de un método. Durante el retardo, el método no ocupa el 1005 de os recurso, sino que está en espera (por ejemplo, esperando una operación de entrada / salida).

Se hace diferencia entra dos tipos de retardos:

- **Retardos embebidos (embedded delays):** Son retardos internos del método. Se modelan asociando un factor de utilización al método. Pueden provocar que la duración de una alternativa sea sobrestimada. Es complicado aprovechar estos retardos si los métodos no son interrumpibles.
- **Retardos externos (external delays**): Están asociado con los NLEs, ya que aparecen cuando una tarea depende de lo que haga otra. Pueden provocar que la duración de una alternativa sea subestimada. En estos casos, el planificador debe propagar la información incierta acerca de los tiempos de inicio.

En ningún caso se considera en qué momento de la ejecución del método ocurre el retardo, ya que esto no se puede predecir (o es muy complicado).

Las duraciones de los retardos se modelan mediante distribuciones discretas de probabilidad.

Incertidumbre

Si bien la incertidumbre es tenida en cuenta en los sliders, esto no es suficiente. Si se desea que el planificador pueda tratar con problemas que involucren incertidumbre, se deben tener en cuenta ciertos aspectos adicionales:

- El cálculo extra requerido
- La propagación de la incertidumbre en la planificación por medio de los NLEs. Por ejemplo, en una relación de habilitación, la incertidumbre de la tarea habilitadora influye en la incertidumbre de la tarea habilitada.
- La posibilidad de evitar planificar tareas que tengan una baja probabilidad de ejecutarse (también debido a los NLEs).
- Poner foco en la generación de alternativas adicionales para un mismo fin, de manera de disponer de mayor redundancia.
- La generación de un plan de contingencia.

Plan de contingencia

La generación de un plan de contingencia implica la capacidad de interrumpir una planificación que no va a lograr los resultados deseados y reemplazarla por otra que sirva para el mismo fin y que pueda ejecutarse en el tiempo disponible.

Un plan de contingencia no consiste meramente en disponer de una alternativa de respaldo. Se debe examinar cuidadosamente el árbol del sistema en estudio para poder aprovechar la estructura del mismo, ubicando posibles puntos de falla y elaborando procesos de recuperación para cada caso.

En especial, se deben detectar los CTERs. Si una planificación atraviesa una CTER, esto se debería tener en cuenta, ya que una falla en este punto podría afectar a la planificación entera. Por ejemplo, podríamos tener do planificaciones que tengan el mismo valor de acuerdo a las metas especificadas por el cliente, pero una tener un CTER cerca del final y otra cerca del principio (podrían, incluso, ser distintos ordenamientos de la misma alternativa). Puede darse el caso de que, si ocurre una falla en la CTER, la primer planificación disponga aún de tiempo para ejecutar una aproximación, mientras que la segunda no disponga del tiempo suficiente.

Aplicaciones

No encontramos muchos casos prácticos de aplicación de Design-to-Criteria. Todos los ejemplos dados en la documentación son casos de búsqueda de información.

Estos algoritmos parecen estar fuertemente enfocados hacia la inteligencia artificial. Sin embargo, la flexibilidad que brinda el hecho de poder especificar los criterios hace que su uso no se limite sólo a este ámbito.

La gran cantidad de cálculo necesaria para realizar cada estimación nos hace pensar que es aplicable a tareas cuyo tiempo de ejecución sea relativamente largo. No serviría, por ejemplo, para problemas cuya naturaleza implique decidir qué ejecutar en cuestión de milisegundos.

AUTOEVALUACIÓN DEL MODULO 3:

Preguntas:

1. El quantum de los procesos varía proporcionalmente a su prioridad mientras no se esté en un Sistema de tiempo compartido. ¿Es esto correcto?. Justifique.

2. “En un S.O. preemptive no se pueden hacer presupuestos sobre el tiempo de corrida de un proceso” (Tanenbaum). ¿Es debido a esta afirmación que el S.O. es el que hace el context swtich, dado que es el único que sabe cuando termina el quantum de tiempo de un proceso? ¿Es esto correcto? Justifique.

3. Explique como funciona la planificación de múltiples niveles.

4. Si la cola de bloqueados se llena, los procesos son enviados automáticamente a la Ready Queue. ¿Es verdad ésto? Justifique.

5. El Short Term Scheduler se ocupa entre otras cosas de balancear la carga de procesos I/O Bound y CPU Bound. ¿Es verdad esto? Justifique.

6.- ¿Cuál es la diferencia entre un Scheduling Preemptive y el Nonpreemptive Scheduling?

7.- Defina brevemente highest-response-ratio-next scheduling.

8.- Defina brevemente feedback scheduling.

9.- Enumere y defina las técnicas de planificación de hilos.

10.- Enumere y defina tres versiones de compartir carga de hilos

11.- ¿Cuál es la diferencia entre una tarea de tiempo real rígida y una de tiempo real flexible?

12.- ¿Cuál es la diferencia entre una tarea de tiempo real periódico y tiempo real aperiódico?

13.- Enumere y defina brevemente cinco áreas generales de requisitos para un sistema operativo de tiempo real.

14.- Enumere y defina brevemente cuatro clases de algoritmos Scheduling de tiempo real.

15.- ¿Qué ítems de información acerca de una tarea podría ser un Scheduling de tiempo real?

Multiple Choice:

1.- Si se realiza una migración del sistema operativo a otro hardware distinto se modifica: a) El algoritmo de planificación para funcionar de la misma manera que antes. b) Las llamadas al sistemas que activan el Job Scheduler. c) La cantidad de registros que intercambia el Dispatcher. d) El Quantum de tiempo e) Las funciones del Dispatcher f) Las funciones del Job Scheduler. g) Todas son verdaderas. h) Ninguna es verdadera	**2.- Si la cola de listos está vacía y se deja en ejecución al Job Scheduler en espera de un proceso sucede que…** a) Cuando termina la entrada salida un proceso y llega otro nuevo debe decidir a quien atiende primero. b) Coloca al proceso Nulo en la cola de Listos y se bloquea. c) Coloca en la cola de listos un Job cuando este llega. d) No puede ingresar nunca ningún proceso. e) Todas las anteriores son correctas. f) Ninguna de las anteriores son correctas
3.- El Dispatcher, cambia su forma de activación: a) Si el algoritmo es Round Robin o FCFS	**4.- Cuando se trabaja con una Planificación Preemptive…** a) Si un proceso pide E/S y ocurre una interrupción de

b) Si el algoritmo es FCFS o SJTF c) Si la planificación es Preemptive o Non Preemtive d) Si el hardware es distinto. e) Todas son verdaderas f) Ninguna es verdadera	Hardware atiende primero la E/S. b) Cuando finaliza un proceso se ejecuta el Dispatcher. c) Un proceso retiene el procesador hasta que solicita una E/S o finaliza. d) Todas son verdaderas. e) Ninguna es verdadera
5.- El Job Scheduler… a) Se activa cuando llega un proceso nuevo b) Se activa cuando termina un proceso. c) Saca los Jobs de la cola de suspendidos y los transforma en procesos d) Solicita los recursos a los administradores del Sistema para crear un proceso nuevo. e) Todas son verdaderas f) Ninguna de verdadera.	**6.- El algoritmo Rund Robin…** a) Funciona con una cola circular y un Quantum de tiempo para todos los procesos. b) Padece Inanición si los procesos no solicitan E/S c) Es poco equitativo para los procesos con ráfagas de CPU menores al Quantum de tiempo d) Puede ser modificado para que un proceso ejecute más seguido. e) Todas son verdaderas f) Ninguna es verdadera
7- Cuando se trabaja con planificación Non Preemptive… a) Si un proceso pide E/S y ocurre una interrupción de Hardware atiende primero la solicitud de E/S. b) El Dispatcher se activa por un System Call o por el timer c) Un proceso retiene el procesador hasta que solicita una E/S o finaliza. d) Todas son verdaderas. e) Ninguna es verdadera	**8. ¿Qué política de planificación de procesos logra un reparto más equitativo del tiempo del procesador?** a) FCFS b) SJF c) Métodos basados en prioridades d) Round Robin e) Prioridades fijas f) Todas las anteriores son correctas. g) Ninguna de las anteriores son correctas
9.- Los Niveles de planificación del S.O. son…. a) Extra largo plazo. b) Largo plazo. c) Medio plazo. d) Corto plazo. e) Planificación preemptive f) Planificación Non preemptive g) Todas son verdaderas. h) Ninguna es verdadera	**10.- Cual de los siguientes objetivos debe cumplir el algoritmo de planificación del Scheduler del S.O. …** a) Maximizar el tiempo de espera de usuarios b) Política parcialmente Justa c) Eficiencia en el uso del S.O. d) Maximizar el número de procesos ejecutados e) Tiempo de respuesta excelente f) Equilibrio en el uso de los recursos. g) Ninguna de las anteriores son correctas
11.- Cuáles de los siguientes factores debilitan el desempeño del scheduler …. a) Las intervenciones del operador. b) Las entradas - salidas lentas. c) El multiplexado de recursos y procesos. d) Las llamadas al sistema. e) Todas las anteriores. f) Ninguna de las anteriores.	**12-¿Cuáles de las siguientes razones pertenecen a la terminación de un proceso?** a) Tiempo limite excedido b) El proceso ha esperado más allá del tiempo máximo especificado para que se produzca cierto suceso. c) El proceso intenta ejecutar una instrucción reservada para el sistema operativo. d) El proceso trata de usar un servicio del S.O. e) Todas las anteriores son correctas. f) Ninguna de las anteriores son correctas
13.- Cuál de estos son algoritmos de planificación dinámica en tiempo real… a) Laxitud b) Planificación con tablas estáticas c) Planificación expropiativa con prioridades estáticas d) Algoritmo monótono de tasa e) Algoritmo del primer límite en primer lugar f) Ninguna de las anteriores.	**14.- Cuáles de los siguientes Parámetros se deben tener en cuenta en la planificación en tiempo real:** a) Tiempo de respuesta mínimo b) Número de tareas. c) Tiempo de retorno d) Periodo de activación. e) Plazo de respuesta f) Tiempo de ejecución máximo. g) Tiempo de respuesta máximo. h) Prioridad.
15.- Cual de los siguientes métodos pertenecen a la planificación de hilos en multiprocesadores y de asignación de procesos … a) Métodos con tablas estáticas b) Compartir carga (Load sharing) c) Métodos expropiativos con prioridades estáticas d) Planificación por grupos (gang scheduling) e) Planificación dinámica del mejor resultado f) Asignación dedicada de procesadores g) Planificación dinámica h) Ninguno pertenece.	**16.- Cual de los siguientes pertenecen a los Algoritmos NON-PREEMPTIVE (sin reemplazo o apropiativos)** a) SJN - (Shortest Job Next) b) Planificación con múltiples colas fijas c) SPN - (Shortest Process Next) d) Round Robin (RR e) Planificación por prioridad f) SRT - (Shortest Remaining Time First) g) HRRN - (High Response Ratio Next) h) Ninguno pertenece.
17.- Cual de los siguientes pertenecen a los Algoritmos PREEMPTIVE (con reemplazo o expropiativos) a) SJN - (Shortest Job Next) b) Planificación con múltiples colas fijas c) SPN - (Shortest Process Next) d) Round Robin (RR e) Planificación por prioridad	**18.- Cuál de los siguientes métodos de planificación se utilizan en algoritmos de tiempo real** a) Métodos con tablas estáticas b) Método de compartir carga (Load sharing) c) Métodos expropiativos con prioridades estáticas d) Método de planificación por grupos (gang scheduling) e) Método de planificación dinámica del mejor resultado

f) SRT - (Shortest Remaining Time First) g) HRRN - (High Response Ratio Next) h) Ninguno pertenece.	f) Método de asignación dedicada de procesadores g) Ninguno pertenece.
19.- Cual de las siguientes funciones funciones pertenecen al Process scheduler … a) Tener actualizados los bloques de control de los trabajos (JCB), b) Identificar usuarios y trabajos ordenados por los usuarios y verificar los permisos. c) Contemplar el máximo aprovechamiento del procesador d) Tener actualizadas las tablas de recursos disponibles. e) Ordenar la cola de Listos con los Jobs entrantes. f) Ocuparse del swappeo de los procesos suspendidos g) Asignar el procesador entre el conjunto de procesos preparados h) Ninguno pertenece.	**20.- Cual de las siguientes funciones funciones pertenecen al middle Term Scheduler …** a) Tener actualizados los bloques de control de los trabajos (JCB), b) Identificar usuarios y trabajos ordenados por los usuarios y verificar los permisos. c) Contemplar el máximo aprovechamiento del procesador d) Tener actualizadas las tablas de recursos disponibles. e) Ordenar la cola de Listos con los Jobs entrantes. f) Ocuparse del swappeo de los procesos suspendidos g) Asignar el procesador entre el conjunto de procesos preparados h) Ninguno pertenece.

Respuestas a las preguntas

1. El quantum de los procesos varía proporcionalmente a su prioridad mientras no se esté en un Sistema de tiempo compartido. ¿Es esto correcto?. Justifique.

Si se toma el algoritmo de planificación Round Robin (original) no es correcto, debido a que el quantum de tiempo es un valor fijo y predeterminado. Ahora, si el Round Robin utilizado posee algunas modificaciones, una de éstas podría ser la de darle a los procesos nuevos un quantum inicial mayor que el normal, o variarlo según algún criterio.

Lo que si puede variar es la cantidad de quantums que se les asigna a cada proceso. También se pueden tener colas dinámicas con distintos quantums, y que los procesos segun su prioridad estén en una cola determinada.

2. "En un S.O. preemptive no se pueden hacer presupuestos sobre el tiempo de corrida de un proceso" (Tanenbaum). ¿Es debido a esta afirmación que el S.O. es el que hace el context swtich, dado que es el único que sabe cuando termina el quantum de tiempo de un proceso? ¿Es esto correcto? Justifique.

No, dado que el sistema opertativo no tiene como saberlo, pues no está ejecutando mientras ejecuta la aplicación.

3. Explique como funciona la planificación de múltiples niveles.

La planificación y administración de trabajos y del procesador se ocupa de la gestión de la ejecución. Para ésto el S.O. usa distintas políticas y mecanismos, siempre con el objetivo de aprovechar al máximo el sistema (minimizar costos y tiempos de espera).

Básicamente, los niveles de planificación se pueden dividir en tres:

1) Largo Plazo o Job Scheduler: El S.O. recibe el trabajo (programa y datos) definido en el nivel anterior, a través de un Software (llamado *Monitor*, residente en el Kernel del S.O.) que lo carga en Memoria Central y crea los procesos con los vectores de estado. Si los recursos solicitados por cada trabajo están disponibles, se los asigna y los coloca en la "cola de listos", sino se encarga de recuperarlos cuando los trabajos se completaron. Organiza la ejecución con un adecuado planeamiento de recursos y asignación de prioridades, para que el trajajo se ejecute ordenada y eficientemente. También controla la cola de listos, decide cuál será el próximo trabajo que se ejecutará, establece una estrategia para el pasaje entre las colas de "bloqueados" a la de "listos", identifica a los usuarios (login) y a sus trabajos, mantiene actualizados los bloques de control de los trabajos (JCB: registros de estados de todos los trabajos en el sistema), las tablas de recursos disponibles, el balance de procesos que son CPU Bound o I/O Bound, etc.
   Se ejecuta con poca frecuencia por ej.: cuando se necesita crear un nuevo proceso, cuando termina, cuando ingresa un usuario al sistema, etc. por esto tiene prioridad máxima para ejecutar.
2) Plazo Medio o Planificador de Swaping: En este nivel el fin es mantener el equilibrio entre los procesos activos e inactivos, en los "sistemas de tiempo compartido" (a diferencia del nivel anterior, que es muy poco usado en éstos sistemas).
   Se encarga del desplazamiento y utiliza varios criterios para tomar las decisiones. Es el que decide sacar de memoria central y llevar a disco (Swap-Out) a los procesos:
   - inactivos
   - activos en estado "bloqueado" temporal o momentaneamente
   - suspendidos

   Cuando desaparecen las causas de los bloqueos, los trae de disco a memoria central (Swap-In), para continuar su ejecución.
   Es llamado en forma periódica para eliminar de la Memoria Central, los procesos que están inactivos hace tiempo.
3) Corto Plazo o Process Scheduler: Es el que decide quién, cuándo, cómo y por cuanto tiempo recibe el procesador un proceso de la *Ready Queue* para ejecutarlo (Context switch). Según el algoritmo que utilize, toma el siguiente proceso de la "cola de listos" y lo pone a ejecutar.
   Está compuesto por dos modulos recidentes en el Kernel: el *Traffic Controller* que se ocupa del manejo de la cola de listos mediante un algoritmo, y el *Dispatcher* o *Switcher* que asigna el uso de la CPU al primer proceso de la cola y lo pone en ejecución. Entre ambos realizan el Context Swich (cambio entre un proceso y otro en el uso de la CPU) y todos los pasos que éste involucra.

4. Si la cola de bloqueados se llena, los procesos son enviados automaticamente a la Ready Queue. ¿Es verdad ésto? Justifique.
No es verdad, debido a que una cola de bloqueados nunca se llena por ser dinámica (hay sistemas operativos que están preparados para una cantidad determinada de procesos. En estos casos las colas pueden ser estáticas, pero tampoco pueden llenarse dado que el S.O. sabe cuántos elementos pueden insertarse en la cola). Además, si sucediera ésto, las reacciones de las aplicaciónes serían impredecibles.

5. El Short Term Scheduler se ocupa entre otras cosas de balancear la carga de procesos I/O Bound y CPU Bound. ¿Es verdad ésto? Justifique.
El short term scheduler sólo se encarga de asignar el próximo proceso a ser ejecutado a la CPU. El balance de carga de procesos I/O bound y CPU bound es realizado por los planificadores de mayor plazo, ya que es necesaria una visión más amplia de la situación.

6.- ¿Cuál es la diferencia entre un Scheduling Preemptive y el Nonpreemptive Scheduling?
Apropiativo: una vez que el proceso pasa a estado de ejecución, continúa ejecutando hasta que termina o hasta que se bloquea en espera de E/S o al solicitar algún servicio del sistema.
NO Apropiativo**:** el proceso que se está ejecutando actualmente puede ser interrumpido y pasado al estado de Listos por parte del S.O. La decisión de apropiarse de la CPU puede llevarse a cabo cuando llega un nuevo proceso, cuando se produce una interrupción que lleva a un proceso Bloqueado al estado Listo o periódicamente, en función de una interrupción del reloj (quantum).

7.- Defina brevemente highest-response-ratio-next scheduling.

Highest Response Ratio Next (HRRN)

Se elige el proceso Listo con un valor mayor de RR (Response Ratio).

$$RR = (m + s) / s$$

Siendo m = tiempo consumido esperando al procesador
s = tiempo de servicio esperado

Tiene en cuenta la edad del proceso, aunque favorece a los trabajos más cortos (un denominador menor produce una razón mayor), el envejecimiento sin que haya servicio incrementa el valor de la razón, de forma que los procesos más largos pasen finalmente primero, en competición con los más cortos.

8.- Defina brevemente feedback scheduling.
Feed-Back (Realimentación)

Penaliza a los trabajos que han estado ejecutando por más tiempo.
Planificación apropiativa y aplicando un mecanismo dinámico de prioridades, cuando un proceso entra por primera vez en el sistema, se sitúa en RQ0. Cuando vuelve al estado Listo, después de su primera ejecución, se incorpora a RQ1. Y ante cada ejecución siguiente, se envía al nivel inferior de prioridad. Dentro de las colas se usa un mecanismo FCFS.
Si un proceso está en la cola de menor prioridad existente, no puede descender, sino que vuelve a la misma cola repetidamente hasta completar su ejecución. Por lo tanto, esa cola se trata con turno rotatorio.
Presenta el problema de que el tiempo de retorno de un proceso largo puede alargarse mucho, o hasta puede ocurrir inanición si llegan regularmente nuevos trabajos al sistema. Para compensar, se puede variar el tiempo de apropiación en función de la cola: un proceso planificado para RQ1 dispone de 1 unidad de ejecución hasta ser expulsado; a un proceso en RQ2, se le permite ejecutar durante 2 unidades de tiempo; y así sucesivamente.
Se puede evitar la inanición promocionando un proceso a la cola de mayor prioridad luego de que haya esperado servicio en su cola actual durante un cierto tiempo.

9.- Enumere y defina las técnicas de planificación de hilos.

Planificación por grupo
Se planifica un conjunto de hilos afines para su ejecución en un conjunto de procesadores al mismo tiempo
Minimiza los intercambios de procesos. Si un hilo de un proceso se está ejecutando y alcanza un punto en el que debe sincronizarse con otro hilo del mismo proceso, el cual se encuentra en Listo, el primer hilo se queda colgado hasta que se pueda realizar un intercambio en otro procesador para traer el hilo que se necesita.

Asignación dedicada de procesador
Es otro método de planificar por grupos. Consiste en dedicar un grupo de procesadores a una aplicación, mientras dure la aplicación. Cuando se planifica una aplicación, se asigna cada uno de sus hilos a un procesador que permanece dedicado a ese hilo hasta que la aplicación termine su ejecución.

Si un hilo de una aplicación se bloquea, el procesador de dicho hilo quedará desocupado, pues no hay multiprogramación de procesadores. Pero la anulación total del intercambio de procesos durante el tiempo de vida de un programa dará como resultado una aceleración sustancial del programa.

Planificación dinámica

El número de hilos de un programa puede cambiar en el curso de una ejecución
Cada trabajo emplea los procesadores de su partición para procesar un subconjunto de sus tareas ejecutables, organizando estas tareas en hilos.
Cuando un trabajo solicita uno o más procesadores, se siguen las siguientes acciones:

- Si hay procesadores desocupados, se usan para satisfacer la petición.
- Caso contrario, si el trabajo que realiza la petición está recién llegado, se le asigna un procesador individual, quitándoselo a algún proceso que tiene más de un procesador asignado.
- Si no se puede satisfacer alguna parte de la petición, queda pendiente hasta que un procesador pase a estar disponible o hasta que el trabajo anule la petición.
- Al liberar uno o más procesadores, se debe explorar la cola de peticiones de procesador no satisfechas, asignar un solo procesador a cada trabajo de la lista que no tenga procesador asignado y luego recorrer nuevamente la lista, asignando el resto de procesadores según un FCFS.

10.- Enumere y defina tres versiones de compartir carga de hilos

Compartición de carga
Los procesos no se asignan a un procesador en particular. Se mantiene una cola global de hilos listos y cada procesador, cuando está ocioso, selecciona un hilo de la cola.
First Come First Server (FCFS)
Cuando llega un trabajo, cada uno de sus hilos se sitúa consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, toma el siguiente hilo listo y lo ejecuta hasta que finalice o se bloque.
Primero el de menor número de hilos
La cola de listos compartida se organiza como una cola de prioridades, en la que la prioridad más alta se le asigna a los hilos de los trabajos con el menor número de hilos sin planificar. Los trabajos de igual prioridad se ordenan según su orden de llegada.
Primero el de menor número de hilos
Con apropiación: la mayor prioridad se da a los trabajos con el menor número de hilos sin terminar. La llegada de un trabajo con un número de hilos menor que un trabajo en ejecución expulsará los hilos del trabajo planificado.

11.- ¿Cuál es la diferencia entre una tarea de tiempo real rígida y una de tiempo real flexible?
Normalmente es posible asociar un plazo a una tarea en particular donde el plazo especifica el instante de comienza o de finalización. Una tarea rígida de tiempo real debe cumplir el plazo, en otro caso producirá Daños no deseados o un error fatal en el sistema.
Una tarea flexible de tiempo real tiene un plazo asociado, que es conveniente, pero no obligatorio, aunque no haya vencido el plazo, aún tiene sentido planificar y completar la tarea.

12.- ¿Cuál es la diferencia entre una tarea de tiempo real periódico y tiempo real aperiódico?
Una tarea aperíodica debe comenzar o terminar en un plazo o bien puede tener una restricción tanto para el comienzo como para la finalización.
Una tarea periódica el requisito se puede enunciar como " una vez por cada período T " o exactamente cada T unidades.

13.- Enumere y defina brevemente cinco áreas generales de requisitos para un sistema operativo de tiempo real.

Los sistemas operativos de tiempo real presentan requisitos especiales en cinco áreas:

- ***Determinismo:*** un S.O. es determinista si realiza las operaciones en instantes fijos y predeterminados o en intervalos de tiempo predeterminados. Cuando compiten varios procesos por los recursos y por el tiempo del procesador, ningún sistema será completamente determinista. En un S.O. de tiempo real, las solicitudes de servicio de los procesos vienen dictadas por sucesos y temporizaciones externas. El determinismo hace referencia a cuánto tiempo tarda un S.O. en reconocer una interrupción.
- ***Sensibilidad:*** se refiere a cuánto tiempo tarda un S.O. en dar servicio a una interrupción, después de reconocerla.
- ***Control del usuario:*** el control del usuario es generalmente mucho mayor en un S.O. de tiempo real que en un S.O. ordinario. En un sistema de tiempo real resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas. El usuario debe poder distinguir entre tareas rígidas y flexibles y especificar prioridades relativas dentro de cada clase.

- ***Fiabilidad:*** es mucho más importante en sistemas de tiempo real que en los que no lo son. Un fallo transitorio en un sistema que no sea de tiempo real puede resolverse simplemente volviendo a iniciar el sistema. En un sistema de tiempo real, las pérdidas o degradaciones del rendimiento pueden tener consecuencias catastróficas.

- ***Tolerancia a fallas:*** hace referencia a la capacidad de un sistema de conservar la máxima capacidad y los máximos datos posibles en caso de fallo. Un aspecto importante a tener en cuenta es la estabilidad. Un sistema de tiempo real estable es aquel que, en caso de que sea imposible cumplir todos los plazos de ejecución de las tareas, cumple los plazos de las tareas más críticas y de mayor prioridad, incluso si no se cumplen los de alguna tarea menos crítica.

14.- Enumere y defina brevemente cuatro clases de algoritmos Scheduling de tiempo real.

Métodos con tablas estáticas:

- Realizan un análisis estático de las planificaciones posibles.
- El resultado del análisis genera un plan que determina, durante la ejecución, cuándo debe comenzar la realización de una tarea.
- La entrada del análisis consta del tiempo periódico de llegada, el tiempo de ejecución, el plazo periódico de finalización y la prioridad relativa de cada tarea.

Métodos expropiativos con prioridades estáticas*:*

- Se realiza un análisis estático, pero no se traza ningún plan.
- El análisis realizado se usa para asignar prioridades a tareas, de forma que se puede emplear un planificador expropiativo con prioridades.
- La asignación de prioridades se encuentra relacionada con las restricciones de tiempo asociadas a cada tarea.

Métodos dinámicos de planificación

- Se determina la viabilidad durante la ejecución (dinámicamente) en lugar de antes de empezar la ejecución (estáticamente).
- Se acepta una nueva tarea para ejecutar sólo si es factible cumplir con sus restricciones de tiempo.

Métodos dinámicos del mejor resultado

- No se realiza ningún análisis de viabilidad.
- El sistema intenta cumplir los plazos y abandona cualquier proceso ya iniciado y cuyo plazo no se haya cumplido.

15.- ¿Qué ítems de información acerca de una tarea podría ser un Scheduling de tiempo real?

En un estudio de los algoritmos de planificación en tiempo real, los distintos métodos de planificación dependen de:

- Si el sistema lleva a cabo un análisis de planificación
- En caso de realizarse el análisis de planificación, si ésta se realiza estática o dinámicamente
- Si el resultado del análisis genera un plan con respecto al cual se expiden las tareas durante la ejecución

Respuestas del múltiple choice.

1.- c.	2.- b.	3.- a, c.	4.- e.	5.- a, c.
6.- a.	7.- a, c.	8.- d.	9.- b, c, d.	10.- d, e,f.
11.- e.	12.- a, b, c.	13.- a, d, e.	14.- b,d,e, f, g, h.	15.- b, d, f, g.
16.- c, e, g.	17.- b, d, f.	18.- a, c, e.	19.- c, g..	20.- f.