2da Parte: Gestión de Procesos

2do Cuatrimestre - 2024

El apunte siempre es modificado y actualizado. Estudiar del apunte del año y cuatrimestre actual

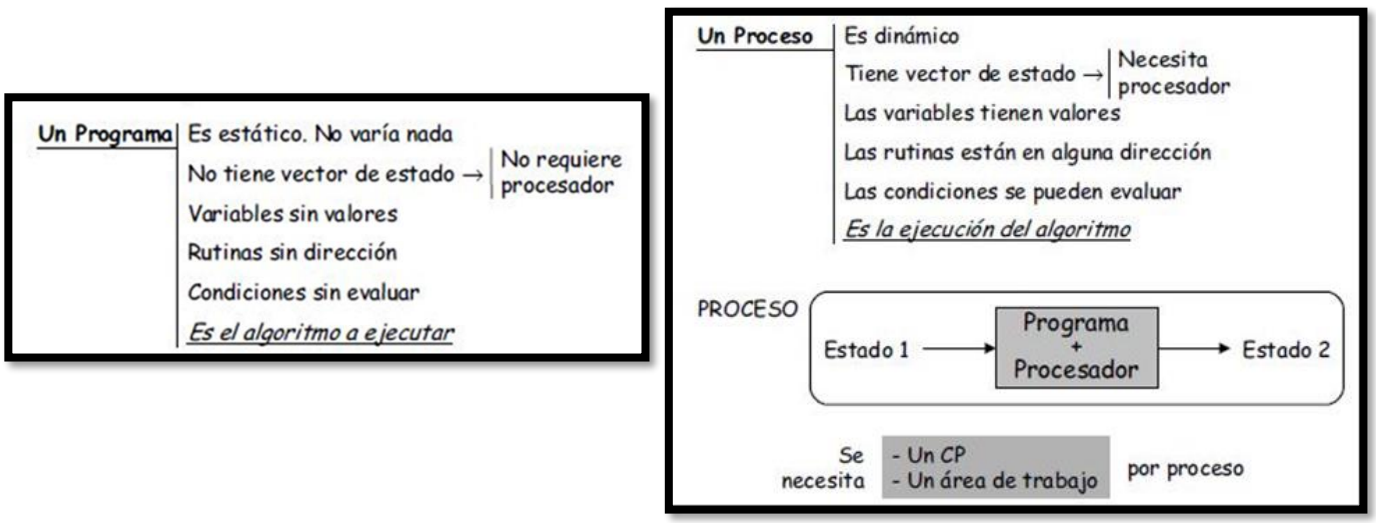
Contenido

2. Gestión de Procesos

El objetivo fundamental de un computador es el de ejecutar programas, por lo que el objetivo principal del sistema operativo será facilitar la ejecución de esos programas. El proceso se puede definir como un programa puesto en ejecución por el sistema operativo y, de una forma más precisa, como **la unidad de procesamiento gestionada por el sistema operativo**.

Un proceso se considera una entidad independiente que está aislada de otros procesos y tiene su propio espacio de memoria, conjunto de registros, identificador único y recursos asignados, como archivos abiertos y sockets de red.

Cada vez que ejecuta un programa en su computadora, se crea un proceso asociado a ese programa. Los procesos permiten que el sistema operativo administre y proteja diferentes instancias de programas y asegure que no interfieran entre sí, necesitará aislamiento y seguridad.

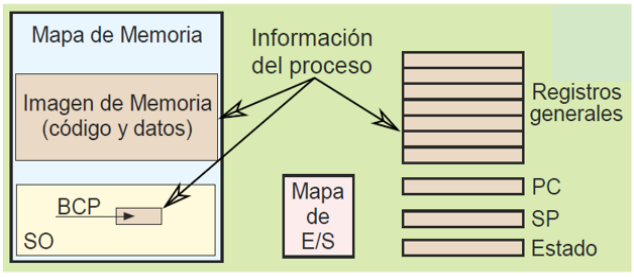


2.1. Introduccion: Contexto de proceso

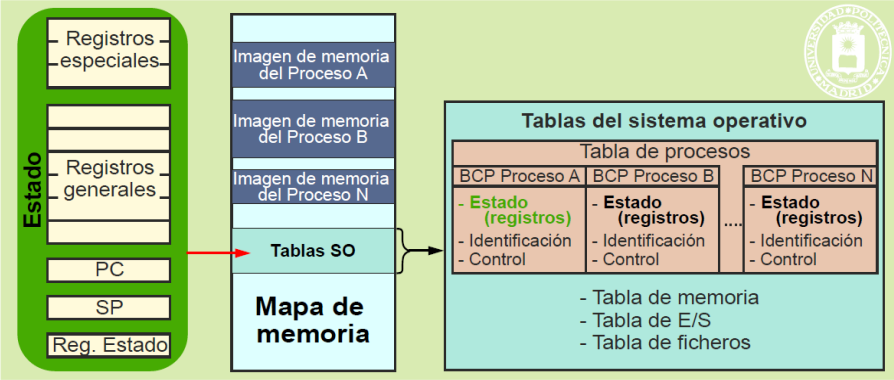
Como ya sabemos, en un sistema multiprogramado se pueden ejecutar simultáneamente varios programas, es decir, en un momento dado puede haber varios procesos. En un sistema con un *único procesador* no se puede decir que haya varios programas ejecutándose estrictamente de una forma simultánea. En realidad, la posesión del procesador se va repartiendo entre los distintos procesos, dando lugar a una ejecución pseudoparalela. No obstante, por simplicidad, nos referiremos, en general, a ejecución simultánea o paralela, tanto si es estricta (*sistema multiprocesador*) como si no lo es (*con un solo procesador*).

Para ejecutar un programa este ha de residir con sus datos en el ***mapa de memoria***, formando lo que se denomina ***imagen de memoria***. Además, el sistema operativo mantiene una serie de estructuras de información por cada proceso, estructuras que permiten identificar al proceso y conocer sus características y los recursos que tiene asignados. Una parte muy importante de estas informaciones se encuentra en el **bloque de control del proceso** (BCP) que tiene asignado cada proceso.

Elementos que constituyen un proceso.



Como se indicó anteriormente, el proceso es la unidad de procesamiento gestionada por el sistema operativo. Para poder realizar este cometido el proceso tiene asociado una serie de elementos de información, *que se resumen en la figura siguiente*, y que se organizan en tres grupos: *estado del procesador*, *imagen de memoria* y *tablas del sistema operativo*.



Es de destacar que el proceso no incluye información de E/S, puesto que ésta suele estar reservada al sistema operativo.

2.1.1. Contexto de un proceso

Se refiere a la información necesaria para mantener y restaurar el estado completo de un proceso en ejecución en un momento dado. Este contexto se guarda en una estructura de datos conocida como Bloque de Control de Procesos (BCP) o Process Control Block (PCB) en inglés. El BCP contiene información sobre el proceso, como su estado, registros, prioridad, ID de proceso (PID), puntero al espacio de memoria asignado, entre otros.

Cuando el sistema operativo cambia de un proceso a otro (por ejemplo, debido a un cambio de planificación o una interrupción), guarda el contexto completo del proceso actual en su BCP antes de cambiar a otro proceso. Esto asegura que cuando el proceso se vuelva a ejecutar, pueda restaurarse desde donde se detuvo anteriormente, con todos sus registros y variables en el estado en que estaban antes de la pausa.

El contexto de un proceso, dependiendo del lugar donde reside la información, está formado por tres tipos de información:

- Bloque de Control de Proceso (BCP)
- Contexto (Imagen) de Memoria
- Contexto (Estado) del Procesador

2.1.1.1. Bloque de Control de Proceso (*o Descriptor de Proceso*)

Es una estructura de datos utilizados por el sistema operativo para mantener información importante sobre un proceso en ejecución. Podemos compararlo con una "ficha" o "tarjeta" que contiene detalles sobre el proceso.

Imagina que tienes varias tareas en tu computadora, como reproducir música, editar un documento y navegar por internet. Cada una de estas tareas es un proceso en el sistema operativo.

Para administrar estos procesos, el sistema operativo crea una "ficha" para cada uno, que es el BCP. Esta ficha contiene información clave sobre el proceso, como su identificador único (PID), estado actual (ejecutándose, en pausa, etc.), los registros del procesador en ese momento, y otros detalles relevantes.

Cuando el sistema operativo necesita cambiar de un proceso a otro (por ejemplo, para darle tiempo de CPU a cada tarea), guarda la ficha del proceso actual con su información completa, incluye los valores de los registros del procesador y la ubicación en la que se detuvo. Luego, cargue la ficha del próximo proceso que debe ejecutarse, de manera que pueda continuar desde donde quedó.

De esta manera, el BCP permite que el sistema operativo administre de manera efectiva múltiples procesos, pausándolos y reanudándolos según sea necesario, para que todos puedan compartir los recursos de la computadora de manera justa y eficiente.

2.1.1.2. Contexto (Imagen) de Memoria

Se refiere a la información y estado relacionados con el uso de la memoria por parte de un proceso en un momento específico. El contexto de memoria incluye detalles sobre la protección y gestión del espacio de memoria virtual que se ha asignado a un proceso en el sistema operativo.

Cuando un proceso se ejecuta, necesita acceder a la memoria para almacenar y manipular datos, ejecutar instrucciones y mantener información relevante para su funcionamiento. El contexto de memoria del proceso almacena información sobre cómo se organiza y administra la memoria para ese proceso en particular.

Algunos elementos importantes que pueden formar parte del contexto de memoria de un proceso son:

- ***Espacio de memoria asignado***: Indica la cantidad de memoria virtual que ha sido asignada al proceso. La memoria virtual es el rango de direcciones de memoria que el proceso puede utilizar y se asigna de manera independiente para cada proceso, consumirá a cada uno una "visión" aislada de la memoria.
- ***Tablas de páginas***: Las tablas de páginas son estructuras de datos utilizadas por el sistema operativo para traducir las direcciones de memoria virtual a direcciones físicas en la memoria RAM. El contexto de memoria incluye información sobre estas tablas, lo que permite al proceso acceder a su espacio de memoria de manera adecuada.
- ***Registros de gestión de memoria***: Estos registros contienen información relevante para el proceso, como límites de memoria, permisos de acceso (lectura, escritura, ejecución) y otras configuraciones relacionadas con la administración de la memoria.

- ***Estado de la memoria***: Informa sobre qué regiones de la memoria están en uso por el proceso y cuáles son libres o reservadas para su uso futuro.

2.1.1.3. Contexto (Estado) del procesador

Se refiere específicamente a los contenidos de los registros y otras estructuras internas del procesador en un momento dado. Los registros del procesador son de memoria interna que almacenan datos temporales y resultados de cálculos mientras se ejecutan las instrucciones de un programa.

Cuando se produce un cambio de contexto de un proceso a otro, el sistema operativo necesita guardar el contexto actual del procesador, que incluye los valores de los registros del procesador y otras estructuras de control, en el BCP del proceso actual. Al guardar el contexto del procesador en el BCP, el sistema operativo puede asegurarse de que el nuevo proceso que se va a ejecutar pueda restaurar su propio contexto de procesador antes de continuar con su ejecución desde donde se detuvo.

Registros accesibles en modo usuario:

- Registros generales. De existir registros específicos de coma flotante también se incluyen aquí.
- Contador de programa.
- Puntero o punteros de pila.
- Parte del registro de estado accesible en modo usuario.

Cuando el proceso no está en ejecución, su estado debe estar almacenado en el bloque de ***control de proceso (BCP).*** Por el contrario, cuando el proceso está ejecutando, el estado del procesador reside en los registros y varía de acuerdo al flujo de instrucciones máquina ejecutado. En este caso, la copia que reside en el BCP no está actualizada. Téngase en cuenta que los registros de la máquina se utilizan para no tener que acceder a la información de memoria, dado que es mucho más lenta que éstos.

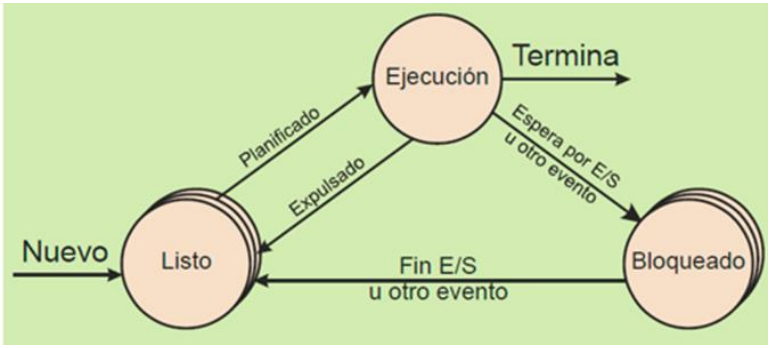
Sin embargo, cuando se detiene la ejecución de un proceso, para ejecutar otro proceso, es muy importante que el sistema operativo actualice la copia del estado del procesador en el BCP. En términos concretos, la rutina del sistema operativo que trata las interrupciones lo primero que ha de hacer es salvar el estado del procesador del proceso interrumpido en su BCP.

Aclaración. No confundir el *contexto del procesador* con el *contexto del proceso*.

El *contexto de un proceso* es una estructura que contiene información sobre el estado completo de un proceso, incluyendo registros, variables y otros detalles. Mientras tanto, el *contexto del procesador* se refiere específicamente a los contenidos de los registros del procesador y otras estructuras internas necesarias para continuar la ejecución del proceso en el punto en que se detuvo. Ambos contextos son fundamentales para el cambio efectivo entre procesos y para permitir la multitarea en sistemas operativos modernos.

2.2. Estados básicos del proceso

No todos los procesos activos de un sistema multitarea están en la misma situación. Se diferencian, por tanto, tres estados básicos en los que puede estar un proceso, estados que detallamos seguidamente:



- **Ejecución**. El proceso está ejecutando en el procesador, es decir, está en fase de procesamiento. En esta fase el estado del procesador reside en los registros del procesador.
- **Bloqueado**. Un proceso bloqueado está esperando a que ocurra un evento y no puede seguir ejecutando hasta que suceda dicho evento. Una situación típica de proceso bloqueado se produce cuando el proceso solicita una operación de E/S u otra operación que requiera tiempo. Hasta que no termina esta operación el proceso queda bloqueado. En esta fase el estado del procesador esta almacenado en el BCP.
- **Listo**. Un proceso está listo para ejecutar cuando puede entrar en fase de procesamiento. Dado que puede haber varios procesos en este estado, una de las tareas del sistema operativo será seleccionar aquel que debe pasar a ejecución. El módulo del sistema operativo que toma esta decisión se denomina **planificador**. En esta fase el estado del procesador esta almacenado en el BCP.

La figura presenta estos tres estados, indicando algunas de las posibles transiciones entre ellos. Puede observarse que:

- solo hay un proceso en *estado de ejecución*, puesto que el procesador solamente ejecuta un programa en cada instante
- el *estado de ejecución* se pasa al *estado de bloqueado* al solicitar, por ejemplo, una operación de E/S (flecha de Espera evento).
- También se puede pasar del *estado de ejecución* al de ***listo*** cuando el sistema operativo decida que ese proceso lleva mucho tiempo en ejecución o cuando pase a listo un proceso más prioritario (*flecha de Expulsado*).
- Del *estado de bloqueado* se pasa al *estado de listo* cuando se produce el evento por el que estaba esperando el proceso (p. ej.: cuando se completa la operación de E/S solicitada).
- Finalmente, del *estado de listo* se *pasa al de ejecución* cuando el planificador lo seleccione para ejecutar.

Todas las transiciones anteriores están gobernadas por el sistema operativo, lo que implica la ejecución del mismo en dichas transiciones

2.3. Cambio de Proceso en Ejecución. Cambio de Contexto

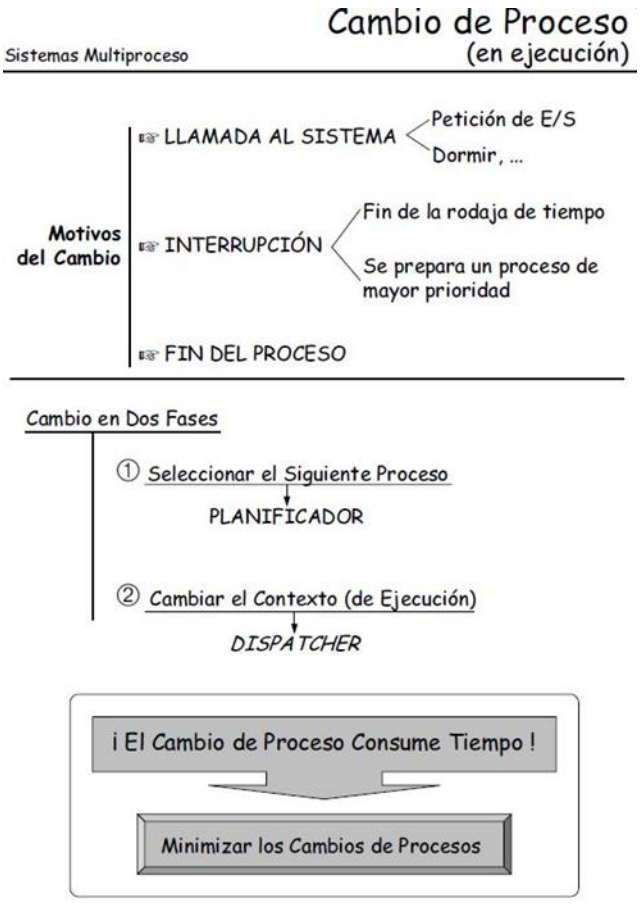
Cuando un proceso en ejecución pierde la posesión de la CPU, se debe a uno de los siguientes motivos:

- Llamada al sistema
- Interrupción
- Fin del proceso

Está claro que ante ciertas llamadas al sistema (*una petición de E/S, dormir, etc*.) el proceso en cuestión queda bloqueado, sin poder continuar la ejecución de instrucciones, hasta que se atienda el servicio solicitado; por esto, cede la posesión del procesador para que pueda ejecutar instrucciones otro proceso que esté preparado.

También puede ocurrir que una interrupción anuncie el final de la actual posesión del procesador. Esto será así si la interrupción es la generada por el reloj del sistema y se detecta que se ha consumido la porción de tiempo asignada (*en un sistema de tiempo compartido*); o bien la interrupción la ha generado algún dispositivo de E/S indicando el fin de una operación previamente solicitada, con lo que el proceso que la había requerido sale de su situación de espera y pasa al estado de Preparado. En un entorno de planificación de CPU por prioridades, si este proceso que acaba de pasar a Preparado tiene mayor prioridad que el que está en Ejecución, se le quita a éste último el procesador para asignárselo al proceso de mayor prioridad.

Por último, el motivo trivial de pérdida de procesador se debe a que el proceso en Ejecución llega a su fin. Detectado esto, el sistema operativo asigna la CPU a algún proceso preparado.

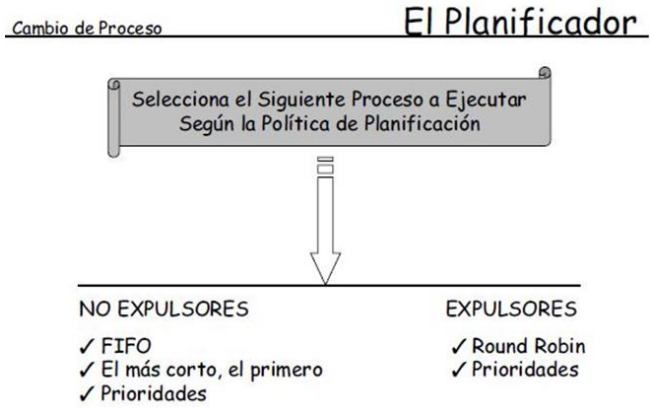


Si un proceso pierde el control de la CPU ¡habrá que asignárselo a otro proceso! Efectivamente, y hay que realizarlo en dos pasos:

1. Seleccionar el siguiente proceso
2. Realizar el cambio de contexto

2.3.1. Planificador (*o scheduler*),

De realizar el primer paso, de seleccionar el siguiente proceso, se encarga el **planificador** (*o scheduler*), que utilizando la política establecida de planificación a corto plazo selecciona un proceso de la cola de Preparados. Esta política de **planificación a corto plazo** establece la asignación de la CPU a uno de los procesos preparados, mientras que la política de **planificación a largo plazo** se utiliza para elegir uno de los trabajos que están en el disco duro esperando para ser cargados en memoria y comenzar su ejecución.



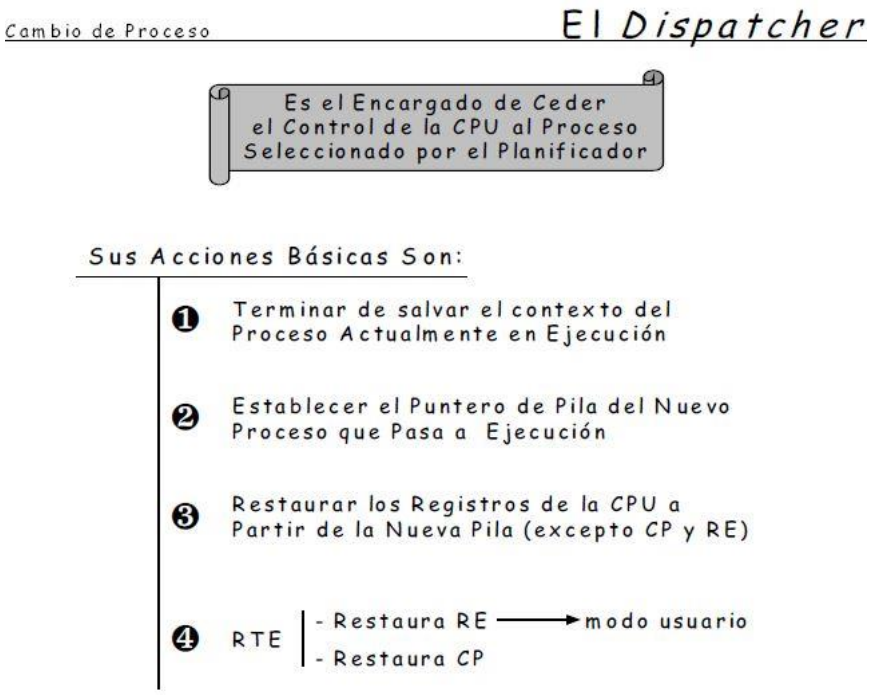
2.3.2. Dispatcher

Una vez conocido el identificador del proceso que va a apropiarse de la CPU, hay que cederle el control del procesador, pero no antes de realizar el cambio de contexto. Para esto, se da control al ***Dispatcher***, que es la parte del sistema operativo que se encarga de terminar de salvar el contexto del programa que abandona la CPU (*o perderlo definitivamente si se debe a una terminación del proceso*) y establecer el contexto del proceso seleccionado. De hecho, el último de los pasos de que consta el cambio completo del contexto, consiste en restaurar el Contador de Programa del proceso elegido. Una vez que el registro Contador de Programa es restaurado, la siguiente instrucción que se ejecuta es ya una instrucción del proceso seleccionado.

Ya hemos visto que el cambio de CPU de un proceso a otro requiere una serie de acciones que se encargan de realizarlas el Planificador y el *Dispatcher*. Pues bien, estas acciones se ejecutan en un tiempo que, desde luego, no es nulo. La elección del proceso a ejecutar suele ser muy simple y no supone una pérdida de tiempo considerable; pero el cambio de contexto sí puede acarrear una gran cantidad de instrucciones, incluidas operaciones de E/S si los contextos de memoria de los procesos que intervienen hay que salvarlos y traerlos, respectivamente, del disco duro, cosa que suele realizarse en los actuales sistemas con Memoria Virtual.

No es deseable, en absoluto, que el tiempo dedicado al cambio de proceso sea considerable. Pensemos en un caso exagerado: si el 20% de las instrucciones ejecutadas en la CPU son las dedicadas a realizar el cambio de ejecución de dos

procesos, esto quiere decir que a las instrucciones de los programas del usuario solamente se le dedica el 80% del tiempo de CPU, mientras que, claramente, lo más deseable es que se acercara lo más posible al 100%.



De la observación anterior se deduce que conviene:

- minimizar el número de cambios de proceso,
- minimizar las operaciones del cambio de contexto.

Esta conclusión la tendremos presente en los siguientes apartados, en los que vamos a tratar más a fondo las acciones concretas del Dispatcher y algunas de las políticas más comunes de planificación de CPU.

Habiendo visto el ciclo de vida de los procesos, sabemos que un proceso que se está ejecutando, eventualmente, por diversos motivos que se han comentado, abandona la posesión del procesador voluntaria o involuntariamente, por lo que hay que cederle la CPU a un proceso que sea lógicamente ejecutable, es decir, que esté Preparado. Si hay más de un proceso Preparado, el sistema operativo debe decidir cuál de ellos se ejecutará primero. La parte del sistema operativo que le corresponde tomar tal decisión es el ***Planificador***, que seleccionará un proceso basado en una política o algoritmo de planificación.

El estado de Preparado se implementa como una cola de procesos dispuestos a ejecutar instrucciones con ayuda de la CPU. Ahora bien, debemos a aclarar que esta cola no tiene por qué ser una cola FIFO (*Primero en Entrar, Primero en Salir*), también podría ser una cola ordenada por prioridades, un árbol o, simplemente, una lista desordenada. El Planificador debe seleccionar uno de los procesos de la lista, pero no basándose necesariamente en la propia estructura de la lista. La estructura de la lista únicamente debe ayudar al Planificador a aplicar el algoritmo de planificación.

2.4. Criterios

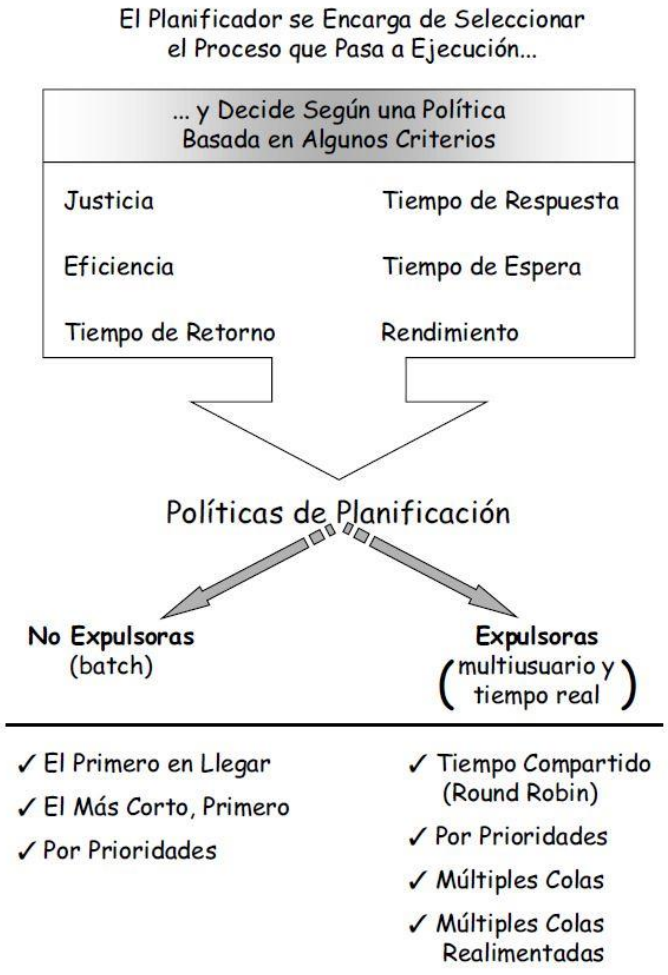
Antes de ver algoritmos concretos de planificación debemos recordar, que la elección que el Planificador va a realizar debe hacerse persiguiendo "el bien del sistema", pues es uno de los cometidos del sistema operativo. No obstante, distintas políticas de planificación tienen diferentes propiedades, y favorecen más a un tipo de procesos que a otros. Antes de elegir el algoritmo a utilizar en una situación concreta, debemos considerar las propiedades de varios algoritmos.

Se han sugerido muchos criterios para comparar algoritmos de planificación de CPU. Decidir las características que se van a utilizar en la comparación es fundamental para la elección del algoritmo más apropiado. Veamos los criterios de comparación más comúnmente utilizados:

- **Justicia**. Cada proceso debe conseguir su porción correspondiente de CPU en un tiempo finito.
- **Eficiencia**. Se debe intentar mantener la CPU ocupada el mayor tiempo posible. Al decir "ocupada" queremos decir ejecutando cualquier proceso que no sea el Proceso Ocioso. En un sistema real, el porcentaje de utilización de CPU suele estar en el rango 40-90%. Una utilización mayor del 90% significaría que el Planificador siempre encuentra procesos en la cola Preparados, o sea, que dicha cola suele contener un número considerable de procesos. En un sistema interactivo, esto puede suponer que los usuarios quizás se están impacientando por obtener las respuestas.
- **Tiempo de Retorno** (*turnaround time*). Desde el punto de vista de un proceso, el criterio más importante es cuánto tiempo se va a necesitar para ejecutarse completamente. El Tiempo de Retorno es la suma de los periodos que se pasan esperando a cargarse en memoria, esperando en la cola de Preparados, ejecutándose en la CPU, y esperando por operaciones de E/S. Así pues, se debe minimizar el tiempo de retorno de un proceso. Afecta principalmente a los procesos batch.
- **Tiempo de Espera**. El algoritmo de planificación de CPU no afecta a la cantidad de tiempo que un proceso se pasa realizando operaciones de E/S, solamente afecta al tiempo que un proceso se pasa en la cola Preparados. El Tiempo de Espera es la suma de todos los momentos que un proceso pasa en la cola de los procesos preparados
- **Tiempo de Respuesta**. En un sistema interactivo, el Tiempo de Retorno puede no ser un buen criterio. A menudo, un proceso puede producir algunos resultados al principio, y puede continuar calculando nuevos resultados mientras los anteriores se le muestran al usuario. Así, tenemos que otra medida es el tiempo que transcurre desde que se le hace una petición al sistema hasta que empieza a responder, sin tener en cuenta el tiempo que se tarda en mostrar la respuesta completa.
- **Rendimiento**. Se debe maximizar el número de trabajos procesados por unidad de tiempo.

Sistemas Multiproceso

Planificación de Procesos



Es deseable maximizar la utilización de la CPU y minimizar los tiempos de retorno, de espera y de respuesta, todo ello con la mayor justicia para todos los procesos. Sin embargo, es fácil observar que algunos de estos criterios son contradictorios. Por ejemplo, para que en los procesos interactivos el tiempo de respuesta sea bueno, se puede impedir que se ejecuten procesos batch por el día, reservando para éstos las horas nocturnas en las que no suele haber usuarios en los terminales. Esto no les sentará muy bien a los usuarios que han encargado trabajos en batch, pues ven que el tiempo de retorno se incrementa. Como en otros aspectos de la vida, lo que beneficia a unos, perjudica a otros; así que, en cualquier caso, no habrá que olvidarse nunca del criterio que hace referencia a la justicia.

En pro de la justicia, en la mayoría de los casos se suelen optimizar los valores medios, no obstante, bajo algunas circunstancias, es deseable optimizar los mínimos o los máximos valores, en lugar de la media. Por ejemplo, para garantizar que todos los usuarios obtienen un buen servicio, lo que se desea es minimizar el tiempo máximo de respuesta.

También se ha sugerido que, en los sistemas interactivos, es más importante minimizar la varianza en el tiempo de respuesta, que minimizar su valor medio. Es preferible un sistema con un tiempo de respuesta razonable y predecible, que otro que aunque por término medio resulte más rápido, sea altamente variable.

2.5. Políticas de Planificación

Las políticas de planificación de procesos son un conjunto de estrategias y reglas utilizadas por el sistema operativo para gestionar y asignar los recursos de manera eficiente entre los diferentes procesos en ejecución. Estas políticas son cruciales para garantizar el uso óptimo de la CPU y otros recursos del sistema, evitando conflictos y garantizando la equidad en el acceso a los recursos por parte de los procesos.

En los diagramas de estados de los procesos (ciclo de vida), vimos que cuando un proceso en Ejecución abandona tal estado pasa a Espera o a Preparado, dependiendo si deja el procesador voluntaria o involuntariamente.

- Si los procesos de un sistema nunca dejan la CPU de forma involuntaria, se dice que la política de planificación de CPU es **no expulsora** o no expropiativa (*non preemptive scheduling*).
- Por el contrario, si pueden perder la posesión del procesador sin solicitarlo, nos encontramos con una planificación **expulsora** o expropiativa (***preemptive scheduling***).

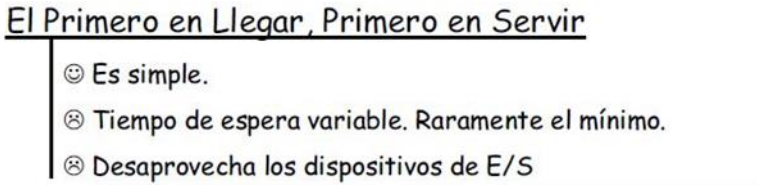
Las políticas no expulsoras suelen aplicarse en sistemas batch o en pequeños entornos ***monousuario***, como el sistema operativo MS-DOS o el entorno Windows-95/98, ambos de Microsoft. Algunos algoritmos que se emplean en esta planificación son *primero en llegar - primero en servir, el más corto primero, y por prioridades*.

Por otra parte, los algoritmos expropiativos se utilizan en sistemas de control industrial y entornos ***multiusuario***. Los algoritmos más utilizados aquí incluyen *Round-Robin, por prioridad*es, de *múltiples* colas y de *múltiples colas realimentadas* (como es el caso de Unix y Windows/NT).

Veamos a continuación algunos comentarios sobre estos algoritmos.

2.5.1. FCFS (*First-Come, First-Served*) o "Primero en llegar, primero en ser atendido

FCFS es uno de los algoritmos de planificación de procesos más simples y directos. En este enfoque, los procesos se ejecutan en el orden en que llegan a la cola de listos. Cuando un proceso llega al sistema, se coloca al final de la cola, y el proceso que está en la parte delantera de la cola es el próximo en obtener acceso a la CPU.



El algoritmo FCFS es fácil de implementar y garantiza un orden de ejecución justo, ya que todos los procesos se ejecutan en el mismo orden en el que llegaron. Sin embargo, presenta una desventaja importante conocida como el "**efecto convoy".** Si un proceso largo y de CPU intensivo se encuentra al principio de la cola, todos los procesos que llegaron posteriormente deberán esperar su turno detrás de este proceso incluso, si son procesos más cortos que podrían completarse rápidamente.

FCFS no puede ser la mejor opción en términos de tiempo de respuesta promedio o rendimiento, especialmente en sistemas con una mezcla de procesos largos y cortos. Para algunos procesos, el tiempo de espera puede ser demasiado largo, lo que resulta en una baja utilización de la CPU y tiempos de respuesta más largos para los procesos que llegaron después.

En general, aunque FCFS es fácil de entender e implementar, otros algoritmos de planificación más avanzados, como SJN (Shortest Job Next) o Round Robin, se utilizan para mejorar el rendimiento y la eficiencia en la gestión de procesos en sistemas operativos modernos. Estos algoritmos tratan de priorizar la ejecución de procesos más cortos o proporcionar una equitativa de la CPU entre los procesos en ejecución, lo que puede conducir a un mejor rendimiento general del sistema.

2.5.2. SJN (*Shortest Job Next*), "El más corto primero"

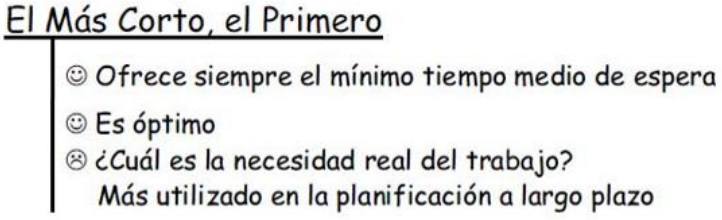
Es un algoritmo de planificación de procesos que da prioridad a los procesos *más cortos* en la cola de listos. En este enfoque, el proceso con el tiempo de ejecución más corto se selecciona para ser ejecutado primero, independientemente del orden en que llegaron los procesos.

El objetivo principal del algoritmo SJN es minimizar el tiempo de espera promedio de los procesos en la cola de listos. Al priorizar los procesos más cortos, aquellos que requieren menos tiempo de CPU se ejecutan primero, lo que reduce la cantidad de tiempo que otros procesos deben esperar para obtener tiempo de ejecución.

Sin embargo, SJN tiene una limitación significativa conocida como "***injusticia para los procesos largos"***. Si hay un proceso largo en la cola de listos, los procesos más cortos que lleguen posteriormente tendrán la oportunidad de ejecutarse antes que el proceso largo. Esto puede generar una situación en la que el proceso largo sufra un tiempo de espera extremadamente largo, lo que se conoce como el problema del **"inversor de carga" (convoy effect)** similar al mencionado anteriormente en FCFS.

El algoritmo SJN es teóricamente óptimo en términos de minimizar el tiempo de espera promedio para una determinada cola de procesos. Sin embargo, en la práctica, puede ser difícil predecir con precisión los tiempos de ejecución de los procesos, y esto puede llevar a una mala estimación y una planificación inadecuada.

En sistemas donde los tiempos de ejecución de los procesos son conocidos o estimados de manera confiable, SJN puede ser una opción viable y puede proporcionar buenos resultados en términos de eficiencia de la CPU y tiempos de respuesta para procesos cortos. En sistemas con una variedad de tiempos de ejecución impredecibles, otros algoritmos de planificación como Round Robin o planificación por retroalimentación pueden ser más adecuados para brindar un rendimiento más equitativo y predecible.



2.5.3. SRT (*Shortest Remaining Time*), “Tiempo Restante más Corto”

Es una variante del algoritmo SJN (Shortest Job Next) y se utiliza para gestionar la planificación de procesos en sistemas operativos. Al igual que SJN, SRT prioriza los procesos con el menor tiempo de ejecución restante para ser ejecutados primero.

La principal diferencia entre SRT y SJN radica en que, en el algoritmo SRT, si un proceso en la cola de listos tiene un tiempo de ejecución más corto que el tiempo restante de ejecución del proceso en ejecución actualmente, se produce una prelación. En otras palabras, si llega un proceso más corto mientras otro proceso está en ejecución, el proceso actual se interrumpe y se otorga la CPU al nuevo proceso más corto.

La prelación en SRT garantiza que los más cortos se ejecuten rápidamente, lo que puede ayudar a reducir el tiempo de respuesta promedio y mejorar el rendimiento del sistema, especialmente en situaciones donde hay procesos cortos que necesitan ser atendidos rápidamente.

No obstante, SRT también puede sufrir un fenómeno llamado **"inversor de carga" (convoy effect)**, similar a SJN, donde un proceso largo podría verse constantemente interrumpido por procesos más cortos, lo que resulta en una mala utilización de la CPU para el proceso largo. Para mitigar este problema, algunos sistemas operativos utilizan técnicas como el envejecimiento (aging), que aumenta la prioridad de los procesos a medida que esperan en la cola de listos durante más tiempo.

En resumen, SRT es una política de planificación dinámica que busca optimizar el tiempo de respuesta de los procesos dando prioridad a aquellos que tienen el tiempo de ejecución restante más corto. La prelación de procesos más cortos sobre los más largos puede mejorar el rendimiento en general, pero es importante considerar el posible efecto de inversor de carga para garantizar un balance adecuado en la planificación de procesos.

2.5.4. Por Prioridades

La planificación por prioridades es un algoritmo de planificación de procesos en el que cada proceso tiene una prioridad numérica que determina su orden de ejecución en relación con otros procesos en el sistema. Los procesos con prioridades más tienen preferencia para obtener tiempo de CPU sobre los procesos con prioridades más altas bajas. Esta política se utiliza para gestionar la preparación de recursos de manera que se puedan satisfacer las necesidades específicas y prioritarias de diferentes procesos.

1. ***Planificación por prioridades estáticas***: En este enfoque, la prioridad de cada proceso se establece de manera estática y no cambia durante su tiempo de ejecución. Los procesos con prioridades más altas se ejecutan primero, y los de prioridad más baja esperan su turno.
2. ***Planificación por prioridades dinámicas***: En este caso, las prioridades de los procesos pueden cambiar dinámicamente durante su ejecución en función de ciertos criterios o eventos. Por ejemplo, un proceso que ha esperado mucho tiempo en la cola de listos puede aumentar su prioridad con el tiempo para evitar el problema de inanición (starvation) y garantizar que todos los procesos tengan la oportunidad de ser ejecutados.

3. ***Prioridad basada en la edad (Aging)***: Es una variante de la planificación por prioridades dinámicas. En este enfoque, los procesos que esperan en la cola de listos durante un período prolongado de tiempo pueden ver aumentada su prioridad a medida que envejecen. Esto evita que los procesos esperen indefinidamente y ayude a equilibrar la impresora de CPU para procesos largos y cortos.

   Dicho concepto de envejecimiento es para evitar problemas de inanición (starvation) y mejorar la justicia en la estimación de la CPU a procesos con diferentes prioridades.

   En esta política, los procesos que esperan en la cola de listos durante un período prolongado de tiempo van aumentando gradualmente su prioridad a medida que envejecen. Es decir, a medida que un proceso pasa más tiempo esperando en la cola de listos sin recibir tiempo de CPU, su prioridad aumenta. Esto significa que incluso los procesos que han estado esperando durante mucho tiempo obtuvieron una oportunidad más justo para ser atendidos, si tienen una prioridad inicialmente más baja.

   El objetivo de la prioridad basado en la edad es evitar situaciones donde un proceso con una prioridad alta acapare constantemente la CPU, dejando a otros procesos con prioridades más bajas esperando indefinidamente. Al aumentar gradualmente la prioridad de los procesos que han estado esperando durante mucho tiempo, se les da una mayor probabilidad de obtener acceso a la CPU y evitar la inanición.

   Esta técnica de envejecimiento es especialmente útil en entornos con cargas de trabajo variables y cambios frecuentes en las necesidades de los procesos. Permite mantener un equilibrio más justo en la protección de la CPU, atendiendo a los procesos más antiguos que han estado esperando, así como a los procesos más nuevos con prioridades más altas.

La planificación por prioridades puede ser tanto expulsora (preemptive) como no expulsora (non-preemptive), dependiendo de si las prioridades son estáticas o dinámicas.

Planificación por prioridades expulsora (Preemptive) En este caso, las prioridades son dinámicas y pueden cambiar durante la ejecución de los procesos. Un proceso en ejecución puede ser interrumpido si llega un proceso con una prioridad más alta. Esto permite que los procesos de mayor prioridad se ejecuten rápidamente y mejore la capacidad de respuesta del sistema. Sin embargo, las interrupciones pueden generar un mayor costo en términos de cambio de contexto (overhead) debido a la frecuencia con la que se detiene y se reanuda la ejecución de los procesos

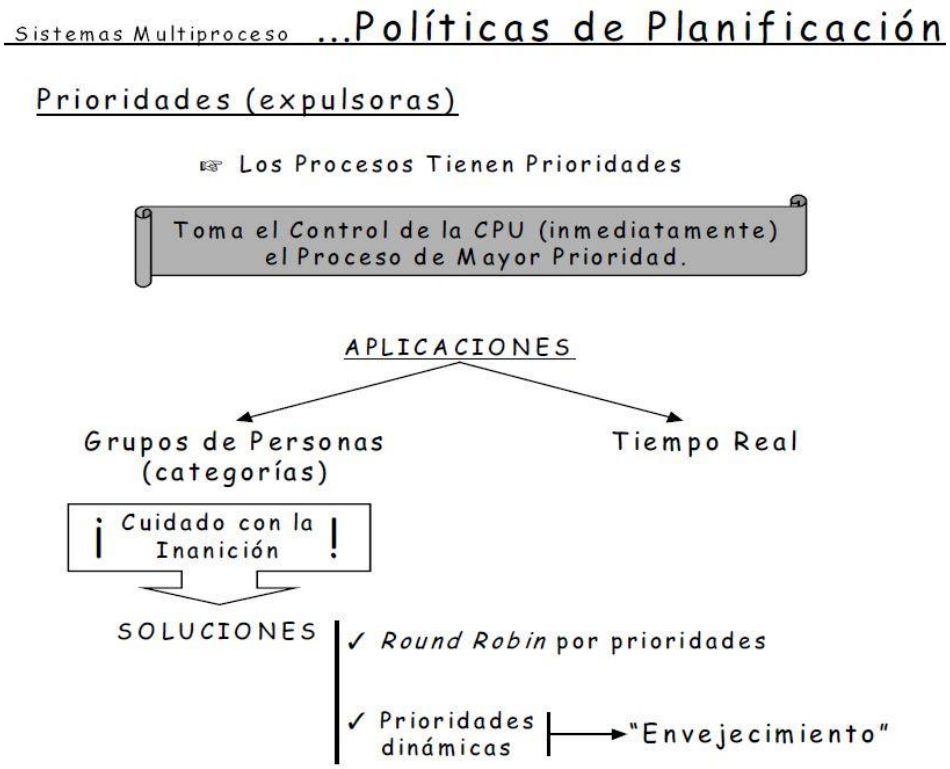
Planificación por prioridades no expulsora (Non-preemptive): En este enfoque, las prioridades son estáticas y no cambian durante la ejecución de los procesos. Un proceso en ejecución no puede ser interrumpido y continuar usando la CPU hasta que finalice su tiempo de ejecución de asignado o realice una operación de E/S. Solo cuando el proceso actual finaliza o se bloquea, se seleccionará el siguiente proceso con mayor prioridad para ejecutarse.

La planificación por prioridades no expulsora tiene la ventaja de ser más simple y conlleva menos sobrecarga de cambio de contexto, ya que no se requiere detener y reanudar con frecuencia la ejecución de los procesos en la CPU. Esto puede ser mejorado en sistemas con procesos predecibles y tiempos de ejecución estables.

Sin embargo, también tiene importantes limitaciones. Si un proceso con baja prioridad está utilizando la CPU y llega un proceso con una prioridad más alta, el proceso real seguirá ejecutándose hasta que termine o realice una operación de E/S, incluso si el nuevo proceso tiene una prioridad más alta y es más urgente. Esto puede conducir a tiempos de respuesta más largos para procesos de alta prioridad y puede afectar el rendimiento general del sistema, especialmente en entornos donde la carga de trabajo es dinámica y las prioridades cambian con frecuencia.

En resumen, la planificación por prioridades no expulsora es más adecuada para sistemas con procesos estables y predecibles, donde el costo de cambio de contexto es un factor importante. Sin embargo, en sistemas con requisitos de capacidad de respuesta más dinámicos, la planificación por prioridad expulsora (preemptive) puede ser más apropiada, ya que permite la interrupción de procesos en ejecución para dar prioridad a aquellos que necesitan ser atendidos rápidamente. La elección de una política de planificación anticipada de los requisitos específicos del sistema y las necesidades de los procesos que se ejecutan en él.

Es importante destacar que la planificación por prioridades puede llevar a cabo la inanición de procesos con prioridades más bajas, ya que los procesos con prioridades más altas recibirán más tiempo de CPU. Para abordar esto, se pueden aplicar técnicas de envejecimiento o limitar el rango de prioridades para garantizar que todos los procesos tengan la oportunidad de ser ejecutados en algún momento. La elección de un algoritmo de planificación por prioridades depende de los requisitos específicos del sistema y las necesidades de los procesos que se ejecutan en él.



2.5.5. Round Robin (RR)

Round Robin (RR) es un algoritmo de planificación de procesos en sistemas operativos que se basa en el concepto de asignar a cada proceso una pequeña cantidad de tiempo de CPU en secuencia, en lugar de ejecutar un solo proceso durante un largo período. El tiempo de CPU asignado a cada proceso se conoce como "*quantum*" o "*cúantum*".

El funcionamiento del algoritmo Round Robin es el siguiente:

1. Todos los procesos en la cola de listos se colocan en una estructura de datos conocida como "cola circular" (circular queue).
2. El proceso que se encuentra en la parte delantera de la cola (el primero en llegar) obtiene una porción del tiempo de CPU, igual al quantum establecido.
3. Si el proceso no ha finalizado su ejecución al final del quantum, se suspende y se coloca nuevamente al final de la cola de listos. De esta manera, los procesos se ejecutan en orden de llegada y se les da oportunidad a todos de tiempo de obtención de CPU.
4. Se continúa este ciclo de ajuste de quantum a cada proceso en la cola de listos. Si no hay procesos en la cola de listos o todos los procesos se ejecutan dentro de su quantum, el algoritmo termina.

El algoritmo Round Robin es especialmente útil en entornos de multiprocesamiento y tiempo compartido, donde varios procesos compiten por la CPU. Al asignar un pequeño quantum a cada proceso, se logra un reparto equitativo del tiempo de CPU, y todos los procesos obtienen oportunidades regulares para ejecutarse. Esto evita que un proceso acapare la CPU por un tiempo prolongado y mejore la capacidad de respuesta del sistema, ya que los procesos son atendidos en intervalos regulares.

Sin embargo, el rendimiento del Round Robin puede verse afectado por el tamaño del quantum. Si el quantum es muy corto, el cambio de contexto (overhead) entre procesos puede ser elevado, lo que afecta la eficiencia. Por otro lado, si el quantum es muy largo, el algoritmo puede comportarse de manera similar a FCFS (Primero en llegar, primero en ser atendido) en términos de tiempos de respuesta. En la práctica, se busca establecer un cuanto que logre un buen equilibrio entre la eficiencia y la capacidad de respuesta para los procesos.

2.5.6. Múltiples colas

Las políticas de planificación de múltiples colas que se utilizan en sistemas operativos para gestionar la ejecución de procesos de manera más eficiente y equitativa. Estas políticas se basan en la idea de clasificar los procesos en diferentes colas de acuerdo con ciertos criterios, y luego se aplica una política de planificación específica para cada cola.

Las políticas de múltiples colas se utilizan para priorizar o clasificar los procesos en función de sus características o prioridades, lo que permite un enfoque más sofisticado en la impresora de recursos de la CPU. Algunas de las políticas de múltiples colas más comunes son:

1. ***Cola de retroalimentación multinivel (MLFQ)***: Es una política de múltiples colas que utiliza varias colas con diferentes prioridades. Los procesos se colocan inicialmente en una cola de nivel alto con una prioridad baja. Si un proceso no se ejecuta completamente en un nivel determinado, se mueve a la siguiente cola de prioridad más alta. De esta manera, los procesos que consumen más tiempo de CPU (procesos largos) descienden en las prioridades y los procesos más cortos pueden obtener tiempo de CPU rápidamente. MLFQ utiliza técnicas de envejecimiento para evitar la inanición de procesos.
2. ***Cola justa ponderada (WFQ):*** Es una política de múltiples colas que asigna pesos a las colas en función de las prioridades o requisitos del proceso. Las colas con pesos más altos reciben una mayor parte del tiempo de CPU, lo que permite que ciertos procesos o tipos de procesos tengan una prioridad más alta en la ejecución.
3. ***Programación Garantizada (GS):*** En este enfoque, cada cola tiene una cuota de tiempo de CPU garantizada. Los procesos en cada cola reciben un tiempo de CPU mínimo asegurado, y si la cola está vacía, la cuota no utilizada se redistribuye a otras colas.
4. ***EDF (Earliest Deadline First):*** No es una política de múltiples colas propiamente dicha, pero se puede implementar como una variante de las políticas de múltiples colas. EDF es una política de planificación en tiempo real que prioriza los procesos en función de sus plazos de ejecución. Los procesos con plazos más cercanos se ejecutan primero para garantizar que se cumplan sus plazos.

Estas políticas de múltiples colas permiten una planificación más flexible y adaptativa de los procesos, lo que puede conducir a un mejor rendimiento y una mayor eficiencia en la reproducción de recursos de la CPU en sistemas operativos con cargas de trabajo diversas y variadas. Cada política tiene sus propias ventajas y desventajas, y la elección de una política específica de las características y requisitos del sistema en particular.

Los ***procesadores actuales*** utilizan una combinación de diferentes políticas de planificación para gestionar la ejecución de procesos y maximizar el rendimiento del sistema. Estos procesadores generalmente operan en entornos multiárea y multihilo, donde múltiples procesos y subprocesos compiten por el tiempo de CPU.

Es importante tener en cuenta que los detalles específicos de la planificación pueden variar según el sistema operativo y la arquitectura del procesador. Además, algunos procesadores modernos también utilizan técnicas de planificación más avanzadas y complejas, como la ***planificación por realimentación*** o ***planificación basada en heurísticas de aprendizaje automático*** para adaptarse dinámicamente a las cargas de trabajo cambiantes y mejorar el rendimiento general del sistema.

2.6. Subprocesos – Tareas - Hilos

2.6.1. Núcleos de un procesador

Los núcleos de un procesador son unidades de procesamiento independientes dentro de un chip de *CPU (Unidad Central de Procesamiento).* Cada núcleo puede realizar tareas de procesamiento de forma autónoma y simultánea, lo que permite a la CPU manejar múltiples tareas a la vez o ejecutar instrucciones en paralelo, lo que se conoce como procesamiento multinúcleo.

Antes de la aparición de los procesadores multinúcleo, los procesadores tenían un solo núcleo que podía ejecutar una sola tarea a la vez. Con el desarrollo de la tecnología y la necesidad de mejorar el rendimiento de las computadoras, los fabricantes comenzaron a incluir múltiples núcleos en una sola CPU.

La presencia de varios núcleos en un procesador permite una mejor utilización de los recursos y aumenta la eficiencia del procesamiento. Por ejemplo, si tiene un procesador de cuatro núcleos, la CPU puede realizar hasta cuatro tareas independientes al mismo tiempo, lo que se traduce en una mejora significativa en el rendimiento general de la computadora, especialmente cuando se realizan tareas que requieren mucho procesamiento, como edición de video, renderizado 3D, juegos o ejecución de múltiples aplicaciones a la vez.

Es importante mencionar que el número de núcleos no es el único factor que influye en el rendimiento de un procesador. Otros elementos, como la frecuencia de reloj, la memoria caché y la arquitectura también juegan un papel crucial en el rendimiento general de la CPU. Sin embargo, los núcleos son un componente clave para entender cómo se distribuye la carga de trabajo y cómo se puede optimizar el procesamiento en un chip de CPU.

2.6.2. Diferencia entre proceso (programa en ejecución) y tareas

- **Programa en ejecución**: Un programa en ejecución es una entidad más amplia que puede incluir varias tareas o hilos en su interior. Es un conjunto de instrucciones y datos almacenados en un archivo ejecutable que se encuentra en el disco duro o en otro medio de almacenamiento. Cuando inicia un programa en su computadora, se carga en la memoria RAM desde el disco duro y comienza a ejecutarse. Este programa puede contener una o varias tareas que realizan diferentes funciones o secciones del programa.
- **Tarea**: Una tarea es una unidad de trabajo más pequeña dentro de un programa en ejecución. Es una secuencia específica de instrucciones que realizan una operación particular en el programa. Una tarea puede ser una función, una rutina o una operación específica que necesita ser realizada. En sistemas operativos modernos, las tareas pueden ser administradas como procesos ligeros (hilos) que se ejecutan dentro del contexto de un proceso.

En resumen:

- Un programa en ejecución es un conjunto de y datos almacenados en un archivo ejecutable.
- Una tarea es una unidad de trabajo más pequeña dentro de un programa en ejecución y puede ser administrada como un proceso ligero (hilo) dentro del contexto del programa.

Por ejemplo, si tiene un programa de procesamiento de imágenes, el programa en ejecución sería la aplicación completa que abriste. Dentro de esa aplicación, podría haber varias tareas o hilos trabajando en diferentes aspectos de la manipulación de imágenes, como cargar una imagen, aplicar filtros y guardar el resultado. Cada una de esas tareas o hilos cumple con una función específica dentro del programa en ejecución.

2.6.3. Hilos (Threads)

El hilo (*también conocido como "thread" en inglés*) es una unidad más pequeña de ejecución dentro de un proceso. Los hilos permiten que un programa realice múltiples tareas de manera concurrente, lo que mejora la eficiencia y el rendimiento del programa en sistemas con múltiples núcleos o procesadores.

La función principal del hilo es ejecutar una secuencia de instrucciones de forma independiente y concurrente con otros hilos dentro del mismo proceso. Cada hilo comparte el mismo espacio de memoria y otros recursos con hilos del proceso, lo que facilita la comunicación y el intercambio de datos entre ellos.

Al utilizar hilos en un programa, se pueden realizar diferentes tareas simultáneamente, lo que puede mejorar la capacidad de respuesta y la eficiencia del programa, especialmente en situaciones donde hay operaciones que pueden realizarse en paralelo.

Algunas funciones clave de los hilos incluyen:

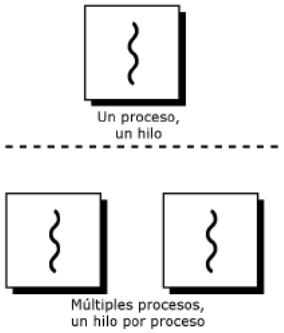
1. ***Multitarea concurrente***: Los hilos permiten que diferentes partes del programa se ejecuten de manera concurrente, lo que puede acelerar la ejecución de tareas y reducir el tiempo de respuesta del programa.
2. ***Compartir recursos***: Los hilos dentro del mismo proceso comparten el mismo espacio de memoria y recursos, lo que facilita la comunicación y el intercambio de datos entre ellos.
3. ***Programación paralela***: Los hilos se utilizan para aprovechar la paralelización en sistemas con múltiples núcleos o procesadores, lo que permite realizar varias tareas al mismo tiempo para mejorar el rendimiento.
4. ***Tareas en segundo plano***: Los hilos se pueden utilizar para realizar tareas en segundo plano mientras el programa principal sigue funcionando, lo que permite la interactividad y la capacidad de respuesta del programa.

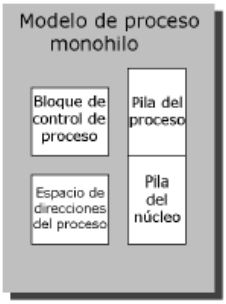
En resumen, los hilos cumplen la función de permitir que un programa realice múltiples tareas de manera concurrente y paralela, compartiendo recursos y mejorando la eficiencia y el rendimiento general del programa en sistemas modernos con múltiples núcleos.

2.6.4. Diferencia monohilo con multihilo

El término multihilo hace referencia a la capacidad de un SO para mantener varios hilos de ejecución dentro del mismo proceso.

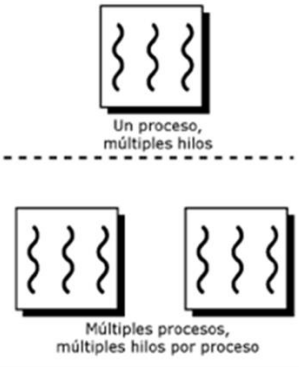
2.6.4.1. Procesos *monohilo* (un solo hilo de ejecución por proceso),

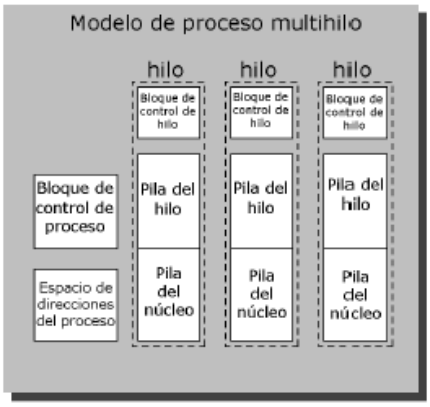




En el monohilo que no existe el concepto de hilo, la representación de un proceso incluye su PCB, un espacio de direcciones del proceso, una pila de proceso y una pila núcleo.

2.6.4.2. Procesos *multihilo*

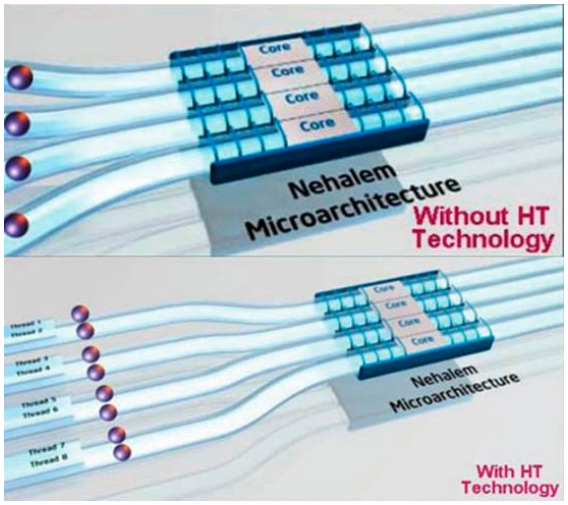




En un SO con procesos ***multihilo***, sólo hay un PCB y un espacio de direcciones asociados al proceso, sin embargo, ahora hay pilas separadas para cada hilo y bloques de control para cada hilo.

2.6.5. *Hyper-Threading (HT)*

Hyper-Threading (HT) es una tecnología desarrollada por Intel que mejora el rendimiento de los procesadores al permitir que cada núcleo físico ejecute múltiples hilos de forma simultánea. Esta tecnología se introdujo por primera vez en procesadores Intel con la serie Pentium 4 y posteriormente se ha aplicado en muchas de sus generaciones de CPU.



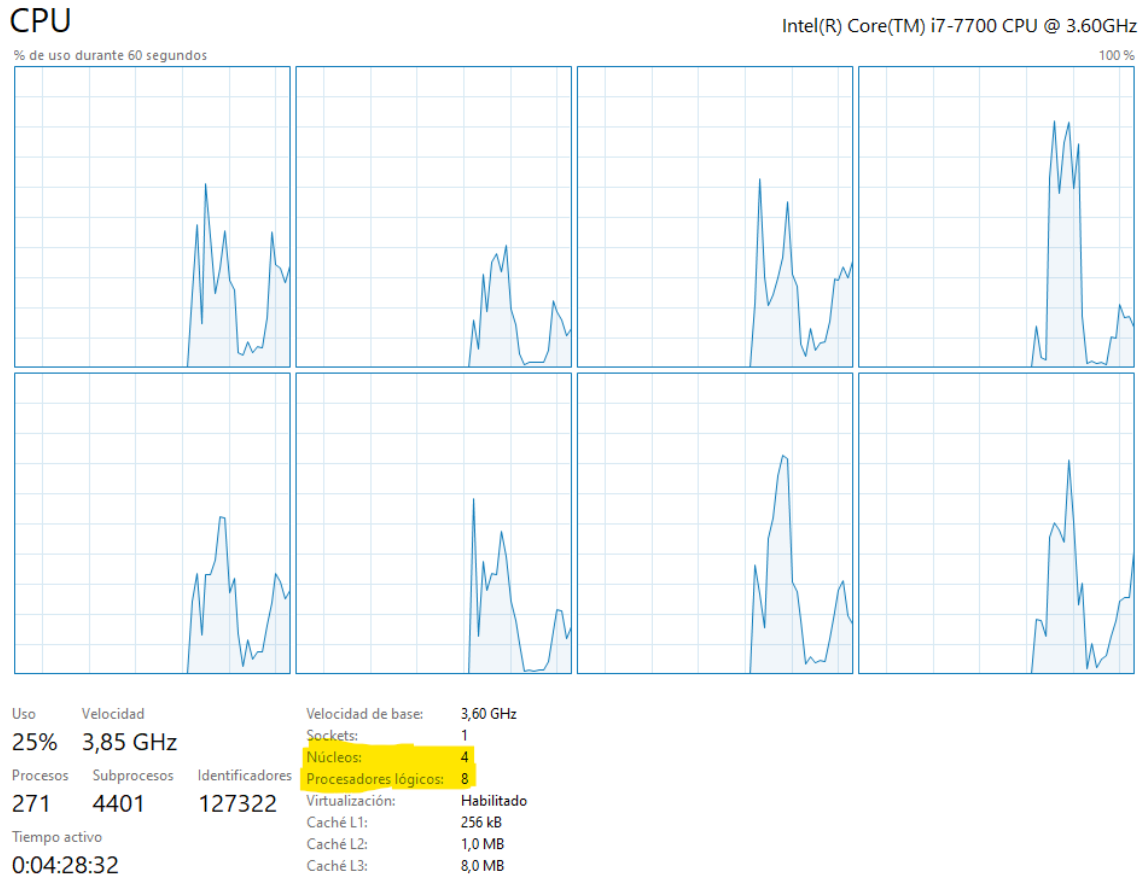
La función principal de Hyper-Threading es crear dos hilos lógicos por cada núcleo físico del procesador. En otras palabras, un procesador de cuatro núcleos con Hyper-Threading tendrá ocho hilos lógicos disponibles para llevar a cabo tareas. Cada hilo lógico se trata como un núcleo virtual y puede ejecutar instrucciones independientes en paralelo.

Es importante tener en cuenta que los hilos lógicos no son núcleos físicos adicionales; en cambio, son recursos compartidos dentro de los núcleos físicos existentes. La tecnología de Hyper-Threading aprovecha el tiempo de inactividad de ciertos componentes del núcleo físico para ejecutar instrucciones de otro hilo, lo que mejora la utilización del procesador y permite realizar más trabajo en un período de tiempo dado.

La tecnología Hyper-Threading puede proporcionar mejoras significativas en tareas que se benefician de la multitarea intensiva, como la virtualización, la edición de vídeo, el renderizado 3D y otras cargas de trabajo pesado. Sin embargo, no todas las aplicaciones se benefician por igual de Hyper-Threading, y en algunas situaciones, no puede haber una mejora significativa o incluso una disminución en el rendimiento debido a la sobreutilización de los recursos del procesador.

Es importante mencionar que AMD, el principal competidor de Intel en el mercado de procesadores, también ofrece una tecnología similar llamada "Simultaneous MultiThreading" (SMT) en sus procesadores Ryzen, que funciona de manera similar a Hyper-Threading de Intel.

Por tanto, si ponemos como ejemplo un procesador i7 de 4 núcleos, este tendrá 8 hilos, y por ello cuando abrimos el Administrador de Tareas de Windows aparecen 8 recuadros en el apartado de CPU (*si sólo aparece un recuadro, hacemos click derecho en la gráfica, y le damos a "Cambiar gráfico a" y seleccionamos "Procesadores lógicos*"). Esto no quiere decir que el procesador tenga 8 núcleos, sino que tiene 8 "núcleos lógicos", y 4 físicos. O lo que es lo mismo, dos hilos por cada núcleo físico.



Bibliografía

- https://www.profesionalreview.com/2019/04/03/que-son-los-hilos-de-un-procesador/
- Silberschatz & P. Galvin ***"Operating Systems Concepts (5th. ed.)"*** Addison-Wesley, 1998
- GALVIN, SILBERSCHATZ; "Fundamento de los Sistemas Operativos"; 7°ed.; Cap. 2; 2006
- Williams Stallings; "***Sistemas Operativos, 2ª ed."*** Prentice Hall, 1997 (original en inglés, de 1995)
- Andrew S. Tanenbaum "***Modern Operating Systems***· Prentice Hall, 1992
- Francisco Aylagas Romero, Elvira Martínez de Icaya Gómez, (Sistemas Operativos1 - Ingeniería de Computadores) Universidad Politécnica de Madrid (UPM) y Escuela Universitaria de Informática (EUI) http://www.dia.eui.upm.es/Asignatu/Sis_op1/Paco/transpaco.htm
- A.S. Tanenbaum: Modern Operating Systems (3rd edition). Prentice-Hall, 2008.
- W. Stallings: Sistemas Operativos. (5ª Edición). Pearson Prentice-Hall, 2005.
- Wikipedia: http://en.wikipedia.org
- KAT/ATC Facultad de Informática UPV/EHU
- http://systope.blogspot.com.ar/2012/05/procesos-e-hilos.html
- https://omicrono.elespanol.com/2017/07/diferencia-entre-nucleos-hilos-procesador/
- https://www.adslzone.net/2017/01/20/asi-funciona-hyperthreading-la-caracteristica-volar-procesador-intel/