Sistemas y Aplicaciones Informáticas

Tema 16. Sistemas Operativos: Gestión de Procesos.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Sistemas operativos en entornos monousuario y multiusuario (ESI 1).

2. Gestión de procesos.

2.1. Programas y procesos. Estructura y recursos de un proceso. Tipos.

- Un programa es un conjunto de instrucciones ejecutables por un ordenador, formado por algoritmos y estructuras de datos que producen un determinado resultado. Un proceso es una unidad de ejecución de un programa con asignación de recursos del sistema.
- Los procesos siempre tienen asociados ciertos recursos del sistema (procesador, memoria, archivos y dispositivos de E/S), que le son asignados en el momento de su creación y durante su ejecución, y es independiente de cualquier otro proceso. Están formados por lo siguiente:
  - Un espacio de memoria principal, en el que se aloja el código del programa en ejecución, la pila (donde se guardan los parámetros y las variables locales de los módulos, y las direcciones de retorno), la sección de datos globales y la sección de memoria dinámica.
  - La actividad actual del proceso, representada por el valor del contador de programa y el contenido de los registros del procesador.
- Los procesos pueden tener distintas relaciones de comunicación entre sí:
  - *Independientes*. No pueden afectar, ni ser afectados por los demás procesos que se ejecutan en el sistema, compiten por el uso de recursos escasos.
  - *Cooperativos*. Pueden afectar o ser afectados por los demás procesos que se ejecutan en el sistema, colaboran entre sí buscando un objetivo común. Cualquier proceso que comparte datos con otro proceso es cooperativo.

2.2. Multiprogramación y tiempo compartido. Gestor de procesos. Funciones.

- Un sistema operativo está formado por un conjunto de procesos que pueden ejecutarse de manera concurrente. Para maximizar la eficiencia del sistema, se emplean dos técnicas:
  - *Multiprogramación*. Consiste en mantener varios procesos independientes de manera simultánea en la memoria principal. El procesador divide su tiempo entre ellos para que siempre haya un proceso en ejecución, evitando la espera por operaciones de entrada/salida.
  - *Tiempo compartido*. Consiste en ejecutar procesos independientes de un conjunto de usuarios simultáneamente, repartiendo el tiempo del procesador entre todos los procesos, de manera que cada usuario puede interactuar con el sistema como si lo tuviera en exclusiva.
- En sistemas monoprocesador nunca habrá más de un proceso en ejecución en cada instante (pseudomultitarea). En sistemas multiprocesador los procesos se repartirán entre los distintos procesadores, pudiendo existir varios procesos en ejecución simultánea (multitarea real) con la posibilidad de que un proceso pueda ejecutarse en varios procesadores (paralelismo). Esto implica la necesidad de una gestión de los procesos desde dos puntos de vista:
  - *Planificación de procesos*. Debe decidirse en cada momento cuál es el proceso que debe ejecutarse y durante cuánto tiempo para maximizar la utilización del procesador.

    - *Concurrencia de procesos*. Deben establecerse mecanismos de sincronización y comunicación entre los procesos, para que éstos se ejecuten según una secuencia temporal conocida que optimice la compartición de los recursos.
- El gestor de procesos es la parte del sistema operativo que asigna recursos a los procesos, permitiendo el intercambio de información entre ellos. Se encarga de lo siguiente:
    - Creación, cambio de estado y destrucción de procesos.
    - Sincronización y comunicación entre procesos.
    - Planificación y desbloqueo de procesos.

2.3. Estados de ejecución. Transiciones. Bloque de control de proceso.

- A medida que un proceso se ejecuta puede pasar por los siguientes estados:
    - *Nuevo*. El proceso se está creando. Para que un programa se ejecute, el sistema operativo debe crear un proceso para él y asignarle un conjunto de recursos.
    - *En ejecución*. El proceso está en la CPU ejecutando instrucciones. En un sistema monoprocesador, sólo puede haber un proceso en ejecución en un instante dado.
    - *Suspendido*. El proceso está esperando a que ocurra un evento, por ejemplo la finalización de una operación de E/S o la recepción de una señal de otro proceso.
    - *Preparado*. El proceso está esperando a que se le asigne tiempo de procesador.
    - *Terminado*. El proceso finalizó su ejecución, por tanto no ejecuta más instrucciones y el sistema operativo le retira los recursos que consume.
- Las posibles transiciones entre los diferentes estados son las siguientes:
    - *Nuevo a preparado*. El sistema acepta un proceso más porque dispone de recursos para ello.
    - *Preparado a ejecución*. El sistema elige uno de los procesos preparados para ejecutarlo.
    - *Ejecución a terminado*. El sistema operativo termina el proceso que se está ejecutando si éste indica que ha finalizado, o si es abandonado o cancelado.
    - *Ejecución a preparado*. El proceso agota su turno de ejecución, cediendo voluntariamente el procesador o siendo interrumpido para atender a otro proceso de mayor prioridad.
    - *Ejecución a suspendido*. El proceso debe esperar a que suceda un evento determinado o a que finalice alguna operación de entrada/salida.
    - *Suspendido a preparado*. Se produce el evento por el que el proceso estaba esperando, o finaliza la operación de entrada/salida.
    - *Preparado o suspendido a terminado*. Un proceso padre puede terminar con un proceso hijo en cualquier momento, o bien, si el padre termina todos sus hijos se pueden terminar.
- El sistema operativo mantiene una estructura de datos denominada bloque de control de proceso (BCP) por cada proceso admitido, agrupándolos en una lista denominada tabla de procesos. Cada uno de los BCP contiene la siguiente información:
    - *Estado actual del proceso y registros de la CPU*: contador de programa, acumuladores, registros índice, punteros de pila y registros generales.
    - *Información de planificación de CPU*: prioridad del proceso, punteros a colas de procesos.
    - *Información de gestión de memoria*: registros base y límite, tabla de páginas o segmentos.

- *Información de contabilidad*: tiempo de CPU, tiempo consumido, número de identificación.
- *Información de estado de E/S*: dispositivos asignados al proceso, lista de archivos abiertos.

2.4. Creación, ejecución, cambio de contexto y finalización de un proceso.

- Los procesos siempre se crean a partir de otros procesos mediante una llamada al sistema durante el curso de su ejecución. El proceso creador se denomina proceso padre y el proceso nuevo se denomina proceso hijo. El hijo recibe un número de identificación, un bloque de control de proceso, un espacio de direcciones de memoria duplicado de la zona de memoria del padre y todos sus archivos asignados. En el nuevo proceso se realiza una llamada al sistema que reemplaza el código por el del programa a ejecutar.
- Una vez creado, el nuevo proceso pasa al estado preparado en espera de ejecución. Al llegar su turno, durante la ejecución puede llevar a cabo lo siguiente:
  - Solicitar una operación de E/S y pasar al estado suspendido.
  - Crear un nuevo proceso y esperar en estado suspendido a que éste termine.
  - Agotar su turno de ejecución o ser desalojado de la CPU como resultado de una interrupción producida por un determinado evento, y pasar al estado de preparado.
- El desalojo de un proceso de la CPU y la reanudación de otro se denomina cambio de contexto. Consiste en guardar el estado del proceso saliente y recuperar los registros del proceso entrante en sus respectivos bloques de control de proceso. El tiempo de conmutación entre procesos es tiempo perdido y debe de ser lo más rápido posible (1-1000 μs). El hardware en ocasiones facilita el cambio de contexto, con sólo cambiar el puntero al conjunto de registros actual.
- Un proceso puede finalizar por dos motivos:
  - Cuando tras ejecutar su última instrucción le pide al sistema operativo que lo elimine utilizando una llamada al sistema (exit).
  - Cuando el padre emite una llamada al sistema para abortarlo por diversas razones, como:
    - El hijo se excedió en la utilización de alguno de los recursos que se le asignaron.
    - La tarea que se asignó al hijo ya no es necesaria.
    - El padre va a finalizar, y el sistema operativo no permite que los hijos continúen.

2.5. Tareas e hilos. Comparación con procesos. Implementación de hilos.

- Una tarea es una entidad sin capacidad de ejecución que sólo posee recursos. Un hilo es cada uno de los flujos independientes de ejecución de un proceso. Un proceso está compuesto de una tarea, la cual puede tener varios hilos de ejecución.
- Las características de los hilos con relación a los procesos son las siguientes:
  - Un hilo comparte bloque de control de proceso, código, datos y recursos del sistema con los demás hilos del proceso, manteniendo como propios el bloque de control de hilo, el contador de programa, los registros de la CPU, la pila y el estado de ejecución.
  - Los estados de ejecución de los hilos y las transiciones entre estados son similares a los de los procesos a los que pertenecen, aunque independientes de éstos. El bloqueo de un hilo puede bloquear todos los hilos del proceso, aún cuando éste se encuentre preparado.
  - Sólo hay un hilo en ejecución en un instante dado. Un hilo dentro de un proceso se ejecuta secuencialmente, y cada hilo puede crear a su vez sus propios hilos hijos.

  - Los hilos no son independientes entre sí. Todos los hilos pueden acceder a todas las direcciones de la tarea, por tanto un hilo puede leer o escribir la pila de cualquier otro hilo.
  - Se necesita menos tiempo en la creación, la finalización y el cambio de contexto de hilos.
- La implementación de los hilos pueden implementarse de dos maneras:
  - *A nivel usuario*. La gestión de hilos la realiza una aplicación y el núcleo no es consciente de su existencia. Pero no se aprovechan los sistemas multiprocesador, ya que el núcleo asigna un proceso a un solo procesador y sólo se puede ejecutar en él un hilo del proceso a la vez.
  - *A nivel kernel*. Todo el trabajo de gestión de hilos lo realiza el núcleo, el cual mantiene la información de contexto del proceso como un todo y la de cada hilo dentro del proceso. Sin embargo, el paso de control de un hilo a otro requiere un cambio al modo kernel.

3. Planificación de procesos.

3.1. Colas de procesos. Distribuidor y planificador. Protección hardware.

- El sistema organiza los bloques de control de proceso agrupándolos en colas de procesos en espera para administrar su ejecución. Existen dos tipos de colas:
  - *Procesos preparados*. Formada por los procesos en espera de tiempo de CPU. El sistema elige de esta cola alguno de los procesos para pasarlo a ejecución. Los procesos nuevos admitidos y aquellos que han agotado su turno de ejecución pasan a esta cola.
  - *Procesos suspendidos*. Contiene los procesos en espera de un evento (E/S o señal de otro proceso). Cuando éste ocurre, todos aquellos que esperan por él pasan a la cola de preparados. Existen varias colas de suspendidos, una asociada a cada evento, para que todos los procesos de la cola asociada pasen en bloque a la cola de preparados.
- El distribuidor es el módulo encargado de asignar la CPU a los distintos procesos. Su función consiste en ejecutar un proceso durante un tiempo determinado, y una vez agotado realizar el cambio de contexto y transferir el control al siguiente proceso de la cola de preparados.
- El planificador es el módulo encargado de organizar las colas de procesos conforme a una determinada política de planificación, separada del mecanismo de planificación (distribuidor).
- Existe un mecanismo de protección hardware para evitar que un proceso de usuario se bloquee y no devuelva el control al sistema operativo. Consiste en establecer un reloj que genera interrupciones a intervalos regulares, lo que provoca una llamada al sistema que permite que el sistema operativo tome el control de nuevo.

3.2. Objetivos de la planificación. Niveles y tipos. Planificación multiprocesador.

- Los objetivos de una buena planificación de procesos son los siguientes:
  - Minimizar el tiempo medio de espera o de retorno de los procesos.
  - Maximizar el número de procesos ejecutados por unidad de tiempo.
  - Mantener el tiempo de respuesta por debajo de un valor máximo.
  - Minimizar la sobrecarga y equilibrar la utilización de recursos.
- Generalmente se identifican tres niveles de planificación:
  - *Nivel alto o largo plazo*. Determina los procesos que serán admitidos en el sistema para su procesamiento. En general, sólo existe en sistemas que admiten procesamiento por lotes.

  - *Nivel medio o medio plazo*. Decide qué procesos se suspenden o se reanudan, y se encarga de guardar los procesos suspendidos en el disco para liberar memoria principal.
  - *Nivel bajo o corto plazo*. Determina a qué proceso en estado preparado se le asigna la CPU.
- Existen dos tipos de planificación:
  - *Planificación apropiativa*. Una vez que se le ha otorgado la CPU a un proceso, le puede ser retirada por el distribuidor. Garantiza buenos tiempos de respuesta en sistemas de tiempo compartido, pero implica una sobrecarga en el sistema por los cambios de contexto.
  - *Planificación no apropiativa*. Después de otorgarle la CPU a un proceso, no le puede ser retirada hasta que él mismo devuelve el control al sistema operativo. Hace más predecibles los tiempos de respuesta, pero no es adecuada para sistemas de tiempo compartido.
- En sistemas multiprocesador la planificación resulta más compleja. Cualquier procesador disponible podrá ejecutar cualquier proceso de la cola común para todos los procesos, evitando que haya procesadores ociosos y procesadores con muy alta ocupación. Exiten dos posibles enfoques: que cada procesador se autoplanifique asegurando que dos procesadores no escojan el mismo proceso y que no quede ningún proceso sin escoger; o bien que un procesador actúe como planificador de los demás procesadores.

3.3. Algoritmos de planificación no apropiativa.

3.3.1. First Come First Served (FCFS).

- Consiste en implementar la cola de procesos preparados como una cola FIFO, es decir, los procesos se atienden por orden de llegada. Suele formar parte de otros tipos de planificación.
- Es el algoritmo más sencillo, pero el más ineficaz. No puede garantizar buenos tiempos de respuesta, y beneficia a los procesos largos sobre los procesos cortos.

3.3.2. Shortest Job First (SJF).

- Se selecciona para ejecutar hasta su finalización el proceso que en ese instante tenga el menor tiempo estimado de ejecución, teniendo en cuenta también a los procesos nuevos que van llegando. Esto requiere un conocimiento preciso del tiempo de ejecución de los procesos, para ello se emplean funciones estadísticas basadas en ejecuciones anteriores.
- Minimiza el tiempo medio de espera de los procesos, aunque existe posibilidad de inanición, ya que los trabajos largos no se ejecutarán mientras haya trabajos cortos.

3.4. Algoritmos de planificación apropiativa.

3.4.1. Shortest Remaining Time (SRT).

- Se selecciona para la ejecución el proceso que tenga el menor tiempo restante de ejecución en cada momento, teniendo en cuenta también a los procesos nuevos que van llegando.
- Es la versión apropiativa del algoritmo SJF. Implica una mayor sobrecarga debido a que el sistema debe mantener un registro de los tiempos de ejecución de los procesos.

3.4.2. Round-Robin.

- Es igual que el algoritmo FCFS, con la diferencia de que cada proceso dispone de un tiempo máximo de ejecución denominado cuanto. Si el proceso sigue ejecutándose al finalizar el cuanto, el distribuidor lo desaloja y el planificador lo incluye al final de la cola de procesos preparados.

- Si el tamaño del cuanto es grande, se comporta como FCFS. Si el tamaño del cuanto es pequeño, el rendimiento del sistema disminuye debido a los constantes cambios de contexto.
- Necesita de un temporizador que controle el cuanto asignado a los procesos y genere una interrupción cuando finalice. Es muy adecuado para sistemas de tiempo compartido.

3.4.3. Colas multinivel y colas multinivel con realimentación.

- Los procesos se clasifican en colas que se gestionan por algoritmos diferentes y se ordenan por niveles de prioridad. La obtención del tiempo del procesador entre las diferentes colas puede ser por orden de prioridad de las mismas, o dividendo el tiempo de la CPU entre ellas.
- En caso de planificación por orden de prioridad, no puede ejecutarse ningún proceso de una cola si las que le preceden en prioridad tienen algún proceso pendiente. Por tanto puede producirse la postergación indefinida de los procesos de baja prioridad.
- Para evitar esto, se observa el comportamiento dinámico de los procesos a lo largo de su tiempo de vida efectiva (realimentación), permitiendo la movilidad de los procesos entre las colas. De esta manera, los procesos que emplean poco el procesador se mantienen en las colas de mayor prioridad y los que lo utilizan mucho se sitúan en las colas de menor prioridad.

4. Interbloqueo de procesos.

4.1. Concepto. Condiciones necesarias para el interbloqueo.

- El interbloqueo consiste en que dos o más procesos adquieren algún recurso necesario para su operación a la vez que esperan a que se liberen otros recursos que retienen otros procesos, llegándose a una situación que hace imposible que ninguno de ellos pueda continuar.
- Para ello, se deben cumplir de forma simultánea las cuatro condiciones siguientes:
  - *Exclusión mutua*. Los procesos utilizan de forma exclusiva los recursos que han adquirido. Si otro proceso pide el recurso debe esperar a que este sea liberado.
  - *Retención y espera*. Los procesos retienen los recursos que han adquirido mientras esperan a adquirir otros recursos que está siendo retenidos por otros procesos.
  - *No expropiación*. Los recursos no pueden quitarse a los procesos que los tienen. Su liberación se produce voluntariamente una vez que los procesos han finalizado su tarea.
  - *Espera circular*. Existe una cadena circular de procesos en la que cada proceso retiene al menos un recurso que es solicitado por el siguiente proceso de la cadena.

4.2. Prevención de interbloqueos. Algoritmo del banquero.

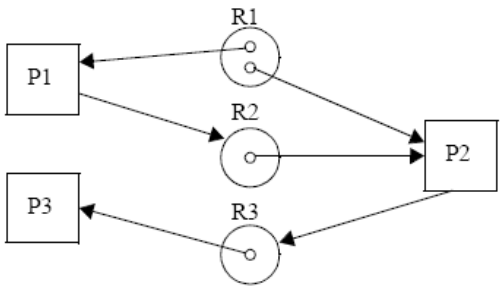
- Para prevenir la aparición de interbloqueos basta con evitar una de las condiciones necesarias para que no se produzca. Otra manera es considerando la posibilidad de que se produzca un bloqueo cada vez que se asigna un recurso. Si se prevé el interbloqueo el recurso no se concede.
- La técnica más utilizada para implementar este método es el conocido como algoritmo del banquero, el cual asegura que el número de recursos asignados a todos los procesos nunca puede exceder del número de recursos del sistema. Nunca se puede asignar recursos de modo que no queden suficientes para satisfacer las necesidades de todos los procesos.
- El algoritmo permite las condiciones de exclusión mutua, de no expropiación y de retención y espera. Los procesos piden recursos al sistema operativo de uno en uno. Una petición que no

conduce a un estado seguro se rechaza y una petición que conduce a un estado seguro se concede. Debido a que siempre se mantiene al sistema en un estado seguro todos los procesos podrán eventualmente conseguir los recursos que necesitan y finalizar su ejecución.

- La gestión de los recursos suele ser conservadora, ya que los estados que realmente producen interbloqueos son un subconjunto de los estados inseguros. Además, el algoritmo requiere que los procesos conozcan por adelantado sus necesidades máximas.

4.3. Detección y recuperación de interbloqueos.

- Para facilitar la detección de los interbloqueos se suele utilizar un grafo dirigido que indica las asignaciones de los recursos a los procesos y las peticiones que estos realizan. Los nodos del grafo son procesos y recursos, y cada arco conecta el nodo de un proceso con el nodo de un recurso. A este grafo se le denomina grafo de asignación de los recursos del sistema.
- De forma gráfica los procesos se representan con cuadrados y los recursos de un mismo tipo con círculos grandes. Dentro de un círculo grande se representa mediante círculos pequeños el número de recursos que hay de ese tipo. La solicitud de un recurso se indica mediante un arco que va desde el nodo proceso solicitante hasta el nodo recurso solicitado; mientras que la propiedad de un recurso se muestra mediante un arco que va del recurso al proceso.



- Si el grafo contiene un ciclo puede existir un interbloqueo. Si sólo hay un elemento por cada tipo de recurso la existencia de un ciclo es una condición necesaria y suficiente. Si cada tipo de recurso tiene varios elementos, una condición suficiente es la existencia de un nudo en el que no hay ningún camino de los que forman el nudo que a su vez no sea ciclo.
- Una vez que se ha detectado el interbloqueo se debe romper para que los procesos puedan finalizar su ejecución y liberar los recursos. Existen varias opciones:
  - Suspender algunos de los procesos bloqueados para tomar sus recursos y reanudarlos.
  - Reiniciar uno o más de los procesos bloqueados.
  - Expropiar los recursos de algunos de los procesos bloqueados.

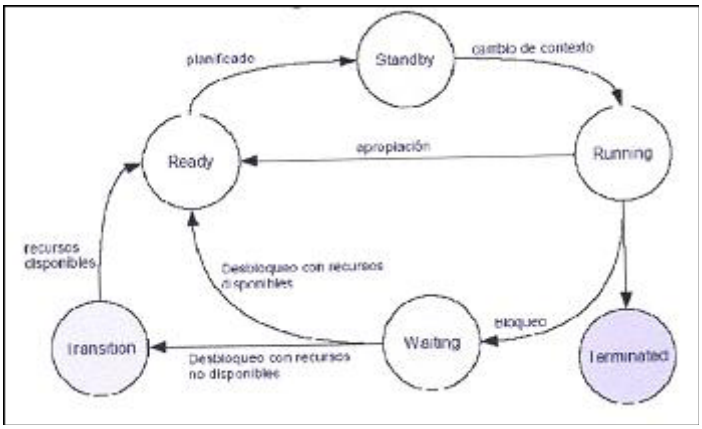
5. Ejemplos de sistemas operativos.

5.1. Gestión de procesos en Windows XP.

- Un proceso es una entidad implementada como un objeto sin estados de ejecución, que posee una serie de recursos y un espacio de direcciones propio y aislado. Para poder ser ejecutado debe tener al menos un hilo, ya que se trata de la unidad de ejecución del sistema. Los hilos pueden crear a su vez nuevos procesos y nuevos hilos, y un proceso no termina hasta que no finalizan

todos sus hilos. En sistemas multiprocesador se realiza multiproceso simétrico, por el cual cualquier hilo de usuario y de sistema puede ser asignado a cualquier procesador.

- Un hilo puede pasar por los siguientes estados: *ready* (listo para ejecutarse), *running* (se está ejecutando), *standby* (seleccionado para ser ejecutado en un procesador particular), *waiting* (suspendido esperando un evento) y *terminated* (finalizado).



- La planificación de la CPU se realiza por colas multinivel que contienen hilos, gestionadas cada una según el algoritmo Round-Robin. Los procesos pueden pertenecer a seis clases posibles de prioridad, mientras que un hilo puede tener cualquier prioridad dentro del rango 0-31, siendo 31 la más alta. La prioridad de un hilo es relativa a la prioridad del proceso al que pertenece, y todos los hilos de la misma prioridad están en la misma cola. La planificación entre las diferentes colas se realiza por orden de prioridad.

5.2. Gestión de procesos en GNU/Linux.

- Linux mantiene una estructura de datos característica de cada proceso denominada *task_struct*, en la que se almacena toda la información relacionada con el proceso. En el campo *state* de la estructura *task_struct* se mantiene el estado de cada proceso:
  - *Task_Running*. Proceso activo que se está ejecutando o está listo para ejecutarse.
  - *Task_Interrumpible*. Proceso en espera que puede activarse por señales o por interrupciones.
  - *Task_Ininterrumpible*. Proceso suspendido que puede activarse sólo por interrupciones.
  - *Task_Stopped*. Proceso detenido momentáneamente por una señal SIGSTOP.
  - *Task_Zombie*. Son procesos que han finalizado antes de que su padre pueda esperar por ellos (llamada al sistema **wait**) para obtener su identificador y poder liberarlos.

- La llamada al sistema **fork** duplica en memoria el contenido del espacio virtual asignado al proceso padre en otro espacio independiente para el proceso hijo. Existe una llamada al sistema exclusiva de Linux y alternativa a fork pero con más opciones denominada **clone**, que dependiendo de los argumentos que le pasemos, el hijo puede compartir el mismo espacio y los mismos descriptores de archivos que el padre, con un comportamiento similar a los hilos de un proceso. Sin embargo, no se manejan estructuras de datos para los hilos diferentes de las de los procesos, por lo que se puede argumentar que Linux no hace diferencias entre hilos y procesos.
- La planificación está basada en la prioridad de cada uno de los procesos. Se distinguen tres clases de procesos que se distinguen por los campos *policy* y *rt_priority* de la *task_struct*:
  - *Procesos de tiempo real con prioridad alta.* Son aquellos con la mayor prioridad.
  - *Procesos de tiempo real con prioridad baja.* Son aquellos con prioridad media.
  - *Procesos de tiempo compartido*. Son los procesos ordinarios con prioridad baja.
- El algoritmo utilizado es Round-Robin. El planificador utiliza en cada proceso la clase, la prioridad y el número de ciclos que le restan por finalizar su cuanto para calcular un valor (*goodness*). El proceso preparado con el valor *goodness* más alto es el siguiente a ejecutar. Si existen varios procesos con el mismo valor *goodness*, se ejecutará primero el que se encuentre más cerca de la cabeza de la lista de procesos preparados.
- Linux tiene soporte para sistemas con varios procesadores en configuración simétrica (SMP o Symmetric Multi-Processing). En este caso se puede repartir la carga entre las distintas CPUs con objeto de aumentar el rendimiento global del sistema. En este tipo de sistemas no tenemos un único proceso en ejecución, podremos tener tantos como procesadores tenga el sistema. Contemplando la posibilidad de trabajar en un sistema SMP, dentro de la *task_struct* de cada proceso se mantiene información relacionada con cada procesador específico.