**Clases de planificación en LINUX y SCHED\_DEADLINE:**

La planificación es el método mediante el cual los hilos, los procesos o los flujos de datos tienen acceso a los recursos, por ejemplo, tiempo de procesador, ancho de banda en la comunicación, etc.

El planificador es el componente del sistema operativo encargado de la planificación. La necesidad de algoritmos de planificación surgió con la aparición de sistemas operativos multi tareas.

* **Planificador completamente justo** (COMPLETELY FAIR SCHEDULE):

Desde la versión 2.6.23 el planificador tradicional de Linux fue remplazado por el CFS. El 80% del diseño de CFS fundamentalmente modela un procesador multi tarea “ideal”.

El CFS busca mantener el balance en el tiempo de procesador que se asignan a los procesos. Cada proceso debe recibir un tiempo equitativo.

***Cuando un proceso está “fuera de balance”, se le asigna tiempo de ejecución en el procesador.***

Para determinar el balance, CFS mantiene la cantidad de tiempo que se le ha asignado a un proceso en lo que llaman “Virtual Runtime”.

***El CFS utiliza unas colas basadas en el tiempo.***

***El proceso con menor “Virtual Runtime” es el más próximo a ser ejecutado.***

* **ARQUITECTURA CFS:**

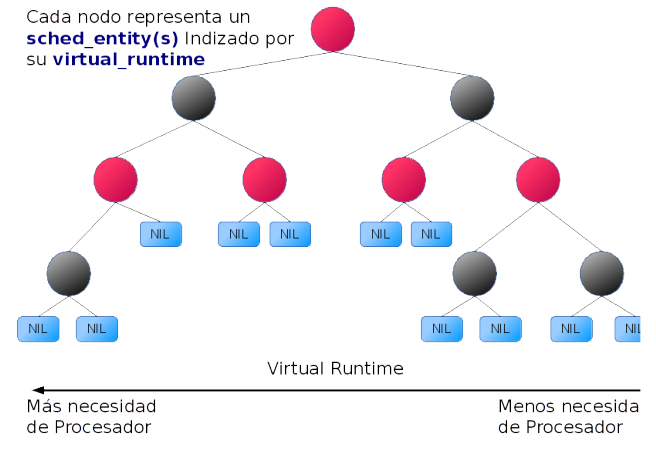
***Un árbol RB esta balanceado.***

***El sub árbol con claves menores a “n” se encuentra a la izquierda.***

***El sub árbol con claves mayores a “n” se encuentra a la derecha.***

***La profundidad de 2 nodos cualquiera no difiere en más de 1.***

***Los sub árboles son balanceados también.***



***Los arboles “rojo-negro” son auto-balanceables. Ningún camino es del doble en tamaño que cualquier otro.***

El nodo más a la izquierda tiene la clave más pequeña. Eso quiere decir que es el nodo con el menor “virtual Runtime”. Es decir, es el nodo que representa al proceso que más necesita ejecutarse.

El nodo demás ala derecha tiene la clave más grande (mayor virtual Runtime). Es el proceso que menos necesita ejecución. Entonces, CFS selecciona el nodo más a la izquierda para ser despachado.

El nodo se elimina del árbol. Si no ha terminado, se inserta de nuevo con un nuevo valor de virtual Runtime.

***Las operaciones en el árbol ocurren en tiempo O(log(n)), donde n es el número de nodos del árbol. De esta manera se pueden ejecutar las operaciones de inserción y eliminación de procesos de manera rápida y eficiente.***

**Proceso de Programación (Process Scheduling):**

Linux como cualquier otro sistema de tiempo compartido, logra el efecto de una ejecución simultanea aparente de múltiples procesos al cambiar de un proceso a otro en un periodo de tiempo muy corto. La programación (Scheduling) se ocupa de cuando tener que cambiar el proceso y que proceso debe elegir.

**Para poder entender un poco más la programación, hablaremos de tres conceptos necesarios, los cuales serían primero, la política de programación (Scheduling Policy) que presenta las opciones hechas por Linux para programar los procesos en el resumen. Segundo, el algoritmo de programación (The Scheduling Algorithm), el cual describe las estructuras de datos utilizadas para implementar la programación y el algoritmo correspondiente.**

* **Política de programación: ()**

El algoritmo de programación de los sistemas operativos tradicionales de Unix debe cumplir varios objetivos conflictivos: rápido tiempo de respuesta del proceso, buen rendimiento para trabajos en segundo plano, evitar la inanición del proceso, la conciliación de las necesidades de los procesos de prioridad baja y alta, y así sucesivamente. El conjunto de reglas que se usa para determinar cuándo y cómo seleccionar un nuevo proceso para ejecutar se llama política de programación .

Procesos interactivos:

Estos interactúan constantemente con sus usuarios y, por lo tanto, pasan mucho tiempo esperando las operaciones de pulsación de teclas y mouse. **Cuando se recibe la entrada, el proceso debe activarse rápidamente, o el usuario encontrará que el sistema no responde. Normalmente, el retardo promedio debe estar entre 50 y 150 ms. La variación de dicho retraso también debe estar limitada, o el usuario encontrará que el sistema es errático**. Los programas interactivos típicos son shells de comando, editores de texto y aplicaciones gráficas.

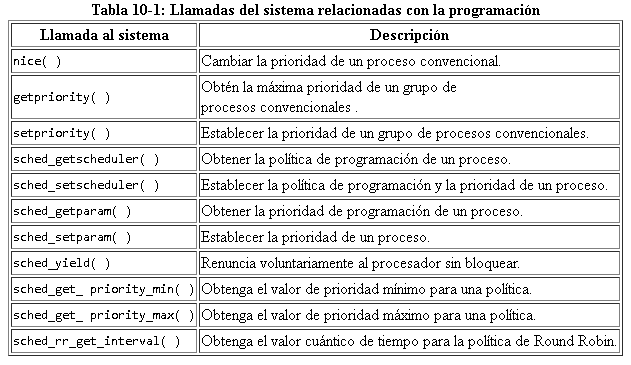
Procesos por lotes:

Estos no necesitan la interacción del usuario y, por lo tanto, a menudo se ejecutan en segundo plano. Dado que dichos procesos no necesitan ser muy receptivos, a menudo son penalizados por el programador. Los programas por lotes típicos son los compiladores de lenguaje de programación, los motores de búsqueda de bases de datos y los cálculos científicos.

Procesos en tiempo real:

Estos tienen requisitos de programación muy fuertes. Tales procesos nunca deben ser bloqueados por procesos de menor prioridad, deben tener un tiempo de respuesta corto y, lo más importante, dicho tiempo de respuesta debe tener una variación mínima. Los programas típicos en tiempo real son aplicaciones de video y sonido, controladores de robot y programas que recopilan datos de sensores físicos.

Los programadores pueden cambiar los parámetros de programación mediante las llamadas al sistema ilustradas en la siguiente Tabla:



* **El algoritmo de programación:**

El algoritmo de programación de Linux funciona dividiendo el tiempo de CPU en épocas. En una sola época, cada proceso tiene una cantidad de tiempo específica cuya duración se calcula cuando comienza la época. **En general, diferentes procesos tienen diferentes duraciones cuánticas de tiempo. El valor cuántico de tiempo es la porción máxima de tiempo de CPU asignada al proceso en esa época.** Cuando un proceso ha agotado su cantidad de tiempo, es reemplazado por otro proceso ejecutable. **La época termina cuando todos los procesos ejecutables han agotado su cuantía; en este caso, el algoritmo del programador vuelve a calcular las duraciones de la cantidad de tiempo de todos los procesos y comienza una nueva época.**

Para seleccionar un proceso para ejecutar, el programador de Linux debe considerar la prioridad de cada proceso. En realidad, hay dos tipos de prioridad:

Prioridad estática:

Este tipo es asignado por los usuarios a procesos en tiempo real y varía de 1 a 99. El programador nunca lo cambia.

Prioridad dinámica:

Este tipo se aplica sólo a los procesos convencionales; es esencialmente la suma del cuanto de tiempo base (que, por lo tanto, también se llama la prioridad base del proceso) y del número de tics de tiempo de CPU que quedan al proceso antes de que su cuanto caduque en la época actual.

**Por supuesto, la prioridad estática de un proceso en tiempo real es siempre más alta que la prioridad dinámica de un proceso convencional: el programador comenzará a ejecutar procesos convencionales solo cuando no haya un proceso en tiempo real en un TASK\_RUNNING estado.**

SCHED\_FIFO

Un proceso de primera entrada, primera salida en tiempo real. Cuando el programador asigna la CPU al proceso, deja el descriptor del proceso en su posición actual en la lista de colas de ejecución. Si no se puede ejecutar ningún otro proceso en tiempo real de mayor prioridad, el proceso continuará utilizando la CPU mientras lo desee, incluso si otros procesos en tiempo real que tienen la misma prioridad son ejecutables.

SCHED\_RR

Un proceso en tiempo real Round Robin. Cuando el programador asigna la CPU al proceso, coloca el descriptor del proceso al final de la lista de colas de ejecución. Esta política garantiza una asignación justa del tiempo de CPU a todos SCHED\_RRlos procesos en tiempo real que tienen la misma prioridad.

SCHED\_OTHER

Un proceso convencional, de tiempo compartido.

El policy campo también codifica una SCHED\_YIELD bandera binaria. Este indicador se establece cuando el proceso invoca la sched\_ yield( ) llamada del sistema (una forma de renunciar voluntariamente al procesador sin la necesidad de iniciar una operación de E / S o ir a la suspensión.

rt\_priority

La prioridad estática de un proceso en tiempo real. Los procesos convencionales no hacen uso de este campo.

**Para SCHED\_RRlos procesos en tiempo real, se utilizan solo para implementar el tiempo compartido. Finalmente, para SCHED\_FIFOlos procesos en tiempo real, no se utilizan en absoluto, porque el algoritmo de planificación considera la duración cuántica como ilimitada.**

* **SCHED\_DEADLINE:**

Evidence Srl, en colaboración con ReTiS Lab de Scuola Sant'Anna, ha diseñado y desarrollado un programador de CPU en tiempo real para Linux.

El proyecto se ha llevado a cabo dentro del proyecto ACTORS FP7.

Presentamos la primera versión del planificador en LKML en septiembre de 2009.

El planificador está disponible por defecto en el kernel oficial de Linux desde la versión 3.14.

* **PORQUE SCHED DEADLINE:**

SCHED\_DEADLINE, en cambio, es un planificador de CPU en tiempo real que:

Permite especificar las restricciones de tiempo de cada tarea;

Se basa en el concepto de reserva de recursos, que permite proporcionar aislamiento entre las tareas en ejecución;

Se basa en algoritmos de programación conocidos (es decir, CBS - Constant Bandwidth Server), de modo que las latencias están limitadas y las restricciones de tiempo están garantizadas.

No hace ninguna suposición restrictiva sobre las características de las tareas, que pueden ser periódicas, esporádicas o aperiódicas.

* **¿Para qué es útil SCHED\_DEADLINE?**

Garantías de tiempo: se garantiza a la tarea una parte determinada del tiempo de CPU, independientemente del comportamiento de las otras tareas que se ejecutan en el sistema; estas garantías son importantes para aplicaciones sensibles al tiempo (por ejemplo, control en tiempo real) que necesitan ejecutar una cierta cantidad de trabajo dentro de una restricción de tiempo.

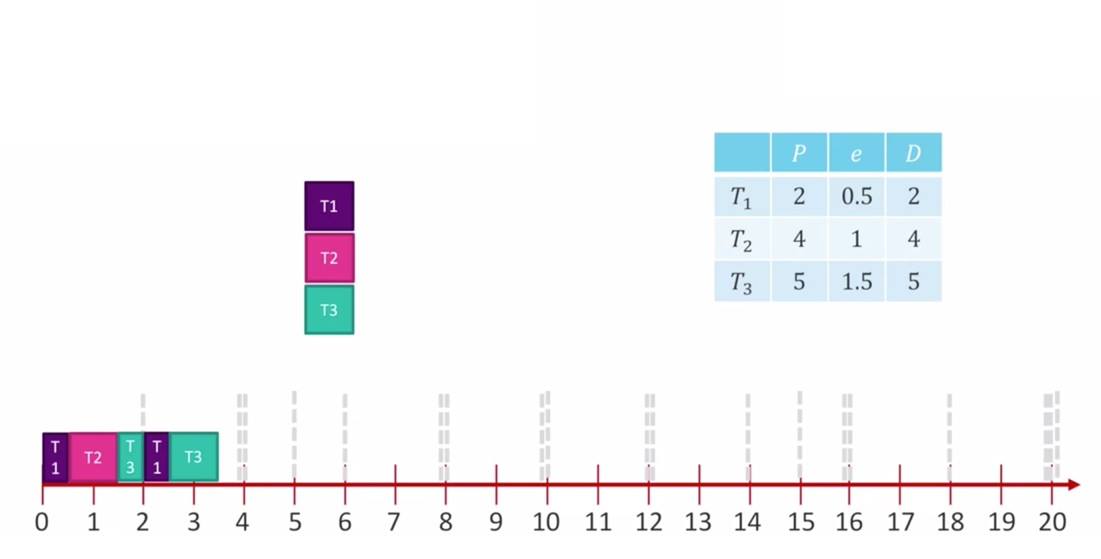
Confinamiento de tiempo: la tarea está obligada a usar no más que su recurso compartido de CPU asignado; esto es útil para limitar la cantidad de CPU utilizada por las tareas de alta prioridad que, en caso de error o cómputo intensivo, pueden dejar de lado al resto del sistema. También es útil en entornos de virtualización, para una asignación justa de la CPU entre varias máquinas virtuales.

* **Earliest Deadline First Scheduler (EDF)**

El EDF es un algoritmo dinámico.

Las prioridades del trabajo se reevalúan en cada punto de decisión.

Reevaluación basada en el deadline relativo de un trabajo.



Cuanto más cerca de la deadline, mayor es la prioridad.

VENTAJAS:

* Muy flexible (los tiempos de llegada y los plazos no se deben conocer antes de la implementación).
* Complejidad moderada.
* Capaz de manejar trabajos aperiódicos.

DESVENTAJAS:

* La optimaliad requiere trabajos preventivos.
* No es óptimo en varios procesadores.
* Difícil de verificar.