UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Sincronización del Tiempo con TSC a nivel del kernel
Autores: Juan Carlos Velasquez Quiroz
Director: Fernando Romero
Codirector: Fernando Tinetti
Asesor profesional:
Carrera: Licenciatura en Sistemas

Resumen

La sincronización del tiempo es importante para diferentes funcionalidades del sistema operativo así como también para ciertos modelos de sincronización de procesos, cuando procesos que interactúan necesitan disponer de un orden de ocurrencia entre sus eventos. Así mismo, esta sincronización puede ser requerida en diferentes ambientes, uno de los cuáles es el ambiente multicore, el cual dispone de un procesador con varios núcleos. Este trabajo tiene por objetivo principal generar un mecanismo para la obtención del tiempo de manera sincronizada en un ambiente multicore bajo el sistema operativo Linux para arquitecturas x86. Para ello se analizaron las funciones que participan en las actividades relacionadas al manejo de tiempo en Linux junto con los dispositivos que son afectados. Este estudio nos permitió entender completamente cómo funciona el sistema del tiempo y a partir de éste, realizando modificaciones, se pudo generar un mecanismo mejorado para obtener una hora sincronizada en un ambiente multicore. Este mecanismo provee como beneficio una mejora con respecto a la resolución y precisión de la solución software actual, que serán comprobados con las diferentes pruebas sobre la solución propuesta. Finalmente, podemos decir que con esta solución software la sincronización del tiempo en un ambiente multicore se ve mejorada.

Palabras Claves

Manejo del tiempo en Linux, Generic Time of Day, Clockevents, time stamp counter, sincronización entre cores, sincronización de relojes en un Sistema Distribuido.

Trabajos Realizados

Análisis de los relojes hardware del sistema.
Análisis de la representación del origen de reloj para la interpolación de tiempo y de las fuentes de eventos.
Análisis del mantenimiento de la hora del día.
Trazado de funciones principales.
Diseño de una escala de manejadores para la sincronización entre cores sin característica constant_tsc.
Pruebas de funcionamiento.
Descripción de beneficios del nuevo reloj.

Conclusiones

El nuevo reloj desarrollado posee menor sobrecarga que el uso del mejor reloj del sistema, mejorándolo en rendimiento y en precisión. Al estar integrado al mecanismo de gestión de energía permite al TSC cambiar de frecuencia. Incluso le permite al TSC detenerse sin necesidad de estar siempre corriendo a máxima frecuencia como en las arquitecturas de CPU con frecuencia constante, justamente permite salvar energía. El uso eficiente de la energía es sumamente importante en cualquier arquitectura.

Trabajos Futuros

Sincronización a nivel de red local.
Integración a la rama principal del kernel.
Comprobación de costo-beneficio entre la solución obtenida y la solución parcial física provista por el microprocesador.
Adecuar la solución para que funcione en ambientes virtualizados y pueda ser utilizado como fuente de reloj.

Fecha de la presentación: SEPTIEMBRE 2014

Universidad Nacional de La Plata

Facultad de Informática

UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE INFORMÁTICA

Sincronización del Tiempo con TSC a Nivel del Kernel

Autor: Velásquez Quiroz Juan Carlos.
e-mail: vela_jc@yahoo.com.ar
Año: 2014.

Director: Romero Fernando.
Co-Director: Tinetti Fernando.

ÍNDICE

RESUMEN

La sincronización del tiempo es importante para diferentes funcionalidades del sistema operativo así como también para ciertos modelos de sincronización de procesos, cuando procesos que interactúan necesitan disponer de un orden de ocurrencia entre sus eventos. Así mismo, esta sincronización puede ser requerida en diferentes ambientes, uno de los cuáles es el ambiente multicore, un procesador con varios núcleos. Este trabajo tiene por objetivo principal generar un mecanismo para la obtención del tiempo de manera sincronizada en un ambiente multicore bajo el sistema operativo Linux para arquitecturas x86. Para ello se analizaron las funciones que participan en las actividades relacionadas al manejo de tiempo en Linux junto con los dispositivos que son afectados. Este estudio nos permitió entender completamente cómo funciona el sistema del tiempo y a partir de éste, realizando modificaciones, se pudo generar un mecanismo mejorado para obtener una hora sincronizada en un ambiente multicore. Este mecanismo provee como beneficio una mejora con respecto a la resolución y precisión de la solución software actual, que serán comprobados con las diferentes pruebas sobre la solución propuesta. Finalmente, podemos decir que con esta solución software la sincronización del tiempo en un ambiente multicore se ve mejorada.

Palabras clave: Manejo del tiempo en Linux, Generic Time of Day, Clockevents, time stamp counter, sincronización entre cores, sincronización de relojes en un Sistema Distribuido.

1. INTRODUCCIÓN

El problema de la sincronización del tiempo tiene importancia cuando se analiza el problema de la sincronización de procesos en la búsqueda de algoritmos para su solución. Podemos definir a la sincronización de procesos como la coordinación de procesos cooperativos que se ejecutan concurrentemente para completar una tarea, con el fin de obtener un orden de ejecución correcto y evitar así estados inesperados.

La sincronización de procesos fue estudiada ampliamente en el campo de la informática. Con tales estudios se pudieron definir dos grandes ambientes que afectan y complican los mecanismos de solución en menor o mayor medida. Uno es el sistema centralizado, en el cual tenemos procesos que se coordinan localmente y el otro es un sistema distribuido, que es una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora [1], en el cual los procesos no necesariamente se encuentran ubicados localmente sino que pueden estar ubicados en diferentes lugares.

Si comparamos ambos ambientes de sincronización nos encontramos con lo siguiente. En un sistema centralizado se hace uso de algoritmos que utilizan mecanismos de coordinación para resolver el problema de la sincronización; la sincronización de procesos en los sistemas distribuidos resulta más compleja que en los sistemas centralizados, debido a que la información y el procesamiento se mantienen en diferentes lugares. Un sistema distribuido debe mantener vistas parciales y consistentes de todos los procesos cooperativos y de cómputo. Tales vistas pueden ser provistas por los mecanismos de sincronización. Estos mecanismos toman la forma de algoritmos distribuidos para sincronizar el trabajo común entre los procesos y estos algoritmos. La necesidad de soluciones diferentes para ambos ambientes se da porque la diferencia más importante entre un sistema distribuido y un sistema centralizado es la comunicación entre los procesos. En un sistema centralizado, la mayor parte de la comunicación supone la existencia de memoria compartida en cambio en los sistemas distribuidos usualmente no existe la memoria compartida [1].

Anteriormente dijimos que muchos algoritmos que se utilizan para sincronizar procesos le prestan gran importancia al tiempo y a la forma de medirlo, por lo que el manejo del tiempo por parte del sistema operativo es muy importante en consecuencia el manejo del tiempo y su sincronización en un sistema distribuido es también importante.

Ahora veamos la sincronización del tiempo en ambos sistemas, pero consideremos un sistema monoprocesador para la comparación. La sincronización en sistemas monoprocesador no requiere ninguna consideración en el diseño del sistema operativo, ya que existe un reloj único que proporciona de forma regular y precisa el tiempo en cada momento. Sin embargo, los sistemas distribuidos tienen un reloj por cada ordenador del sistema, con lo que es fundamental una coordinación entre todos los relojes para mostrar una hora única. La sincronización no es trivial, porque se realiza a través de mensajes por la red, cuyo tiempo de envío puede ser variable y puede depender de muchos factores, como la distancia, la velocidad de transmisión o la propia saturación de la red. Pero la sincronización no tiene por qué ser exacta, y bastará con que sea aproximadamente igual en todas las computadoras con un error conocido.

Como vemos, tenemos dos ambientes diferentes bien diferenciados, pero en la actualidad los sistemas monoprocesador evolucionaron a sistemas con un solo procesador físico y varios núcleos internos llamados multicore. La tecnología multicore es el término que describe al día de hoy los procesadores que tienen dos o más CPUs que son unidades que leen y ejecutan instrucciones de programa. Las instrucciones son instrucciones ordinarias de la Unidad Central de Procesamiento. Esto afecta considerablemente al manejo del tiempo del sistema operativo. En este trabajo pretendemos obtener una mejora de la solución software actual al problema de la sincronización del tiempo en un ambiente multicore sobre arquitecturas x86 basándonos en el sistema operativo Linux.

El análisis del subsistema de manejo del tiempo de Linux nos dio la base principal para poder entender cómo podíamos encontrar una posible solución al problema de la sincronización del tiempo. En este sentido notamos la evolución en complejidad del subsistema así como los inconvenientes encontrados durante el proceso de mejora. Esta complejidad en términos de software provee eficiencia (mejora del código en cuanto a detección de patrones de uso, eliminación de redundancias y optimización de llamadas a funciones) y flexibilidad (incorpora una interface para el manejo de los diferentes dispositivos de reloj dentro del sistema operativo). Con este subsistema se pueden manejar diferentes relojes (ya sean de software o de hardware). El cambio del subsistema del manejo del tiempo fue debido a la evolución de los diferentes dispositivos comprometidos en el manejo del tiempo y en gran medida a las modificaciones sobre las micro-arquitecturas de las Unidades Centrales de Procesamiento modernas. Posteriormente, a través de este análisis notamos cuáles son las limitaciones de la solución software que viene incorporada en el sistema operativo y también cómo podríamos generar una solución alternativa con mejoras con respecto a dicha solución. Finalmente, la solución será comprobada con diferentes pruebas de comparación de funcionamiento en un ambiente real.

1.1. Objetivo de la tesina

El objetivo principal de esta tesina es la obtención de un mecanismo mejorado para obtener el tiempo del sistema logrando el sincronismo de hora en un ambiente multicore.

Los procesadores multicore se han conceptualizado en torno a la idea de ser capaces de hacer posible la computación paralela (más volumen de trabajo que se puede hacer al mismo tiempo). Esto nos permite reducir la potencia y el consumo de calor del sistema sin dejar de ser capaces de mejorar el rendimiento del sistema eliminando el aumento del consumo de energía. Entonces obtenemos más rendimiento con menos o con la misma cantidad de energía. Por ejemplo los fabricantes como Intel y AMD se centraron más en aumentar el rendimiento de la computación y el procesamiento sin aumentar la velocidad de reloj, de esta forma evitan la necesidad de consumir más energía. Podemos concluir que en un ambiente multicore el sincronismo de hora hace referencia a que independientemente de quién esté ejecutando el proceso que obtiene la hora del sistema esta hora siempre debe tener un crecimiento monotónico.

1.2. Estructura de la tesina

Para lograr el objetivo planteado se ha estructurado la tesina de la siguiente manera:
Esta tesina esta organizada en 5 capítulos, además de esta introducción y 7 anexos.

El capítulo 2 esta dedicado a definir los dispositivos físicos que interactúan con el subsistema de manejo del tiempo y como el subsistema los utiliza.
En el capítulo 3 revisaremos un poco de historia con respecto al problema de la sincronización del tiempo en sistemas distribuidos. Analizando los algoritmos principales que abarcan a este problema.
En el capítulo 4 nos centrarnos en los problemas específicos del uso del mejor reloj del sistema como posible origen del reloj en ambientes multicore, así como también se definirá el esquema actual que mantiene los módulos que caracterizan al subsistema del manejo del tiempo.
En el capítulo 5 analizaremos las funciones principales que intervienen en el manejo del tiempo.
Una vez que hayamos identificado las diferentes partes que participan en el manejo del subsistema del tiempo e identificado y comprendido su funcionamiento, en el capítulo 6 trabajaremos en la construcción de un nuevo esquema para obtener un mejor mecanismo para la obtención de la hora, el mismo será refinado para un funcionamiento optimo a medida que se avance hacia una mejor solución.
En el capítulo 7 se proveerá de pruebas de funcionamiento y se mostrarán los beneficios del nuevo reloj.
Para finalizar, en el capítulo 8 daremos las conclusiones de este trabajo y describiremos las posibles alternativas a seguir relacionadas con este trabajo.

El conjunto de 7 anexos complementan el material central de esta tesina, disponible en el CD que acompaña a este trabajo.

2. DESCRIPCIÓN DE DISPOSITIVOS

En este capítulo comenzaremos describiendo cuáles son los dispositivos físicos que el subsistema del manejo del tiempo controla durante su funcionamiento. Estos dispositivos físicos son utilizados para obtener las medidas del tiempo, compuestos por varios circuitos hardware basados en osciladores de frecuencia fija y contadores, los mismos conformarán los llamados relojes hardware.

2.1. Relojes Hardware

En la arquitectura 80x86, el núcleo debe explícitamente interactuar con varios tipos de circuitos de relojes y temporizadores. El circuito de reloj se utiliza para hacer un seguimiento de la hora actual del día y hacer precisas las mediciones del tiempo. Los circuitos temporizadores son programados por el núcleo, de manera que ellos emitan interrupciones en una frecuencia fija y predefinida. Tales interrupciones periódicas son importantes para implementar los temporizadores de software utilizados por el núcleo y los programas de usuario. Los relojes hardware se detallarán a continuación y son los siguientes: Reloj de Tiempo Real, Time Stamp Counter, Temporizador de Intervalos Programable, Temporizador del LAPIC, Temporizador de Eventos de Alta Precisión y el Temporizador de Administración de Energía.

2.1.1. Reloj de Tiempo Real

Todas las PCs incluyen un reloj llamado Reloj de Tiempo Real (también conocido como RTC, reloj hardware o reloj CMOS), que es independiente de la CPU y de todos los otros chips. Este reloj de hardware esta incluido en un circuito integrado, que mantiene la hora actual. Los RTCs tienen una fuente de alimentación alternativa, por lo que pueden seguir midiendo el tiempo mientras la fuente de alimentación de la PC no este disponible (marcan incluso cuando la PC esta apagada). Esta fuente de alimentación alternativa es normalmente una batería de litio en los sistemas antiguos. Pero algunos sistemas nuevos usan un supercapacitor [2], porque son recargables y pueden ser soldados. La mayoría de los RTCs usan un oscilador de cristal [3], pero algunos usan la frecuencia de la fuente de alimentación.

El RTC fue introducido en los PCs compatibles por IBM PC/AT en 1984, cuando los mismos usaron un RTC Motorola MC146818. Luego, se utilizaron RTCs compatibles en las computadoras antiguas, las mismas pueden ser identificadas en las placas base por su distintiva batería negra y por su logo serigrafiado. Actualmente los sistemas nuevos llevan el RTC integrado en el puente sur del chipset de la placa madre [4].

El RTC puede ser utilizado para proporcionar información de tiempo (hora, minuto y segundos), fecha (año, mes y día) y generar interrupciones. Además, dispone de la función de temporizador/contador y de alarma con lo que se podría programar al RTC para activar una interrupción cuando el contador alcanza un valor específico y también para emitir interrupciones periódicas. Estas interrupciones son reportadas por la línea física de interrupción IRQ 8.

Linux y el RTC:

El comportamiento del Linux con respecto al RTC es el siguiente:

En el momento de arranque de la PC el sistema operativo establece el horario del reloj del sistema desde el reloj del hardware (hora y fecha). Para ello utiliza el programa hwclock para copiar el tiempo del reloj del hardware al reloj del sistema. Luego, en el momento de apagarse la PC, se ajusta el horario del reloj del hardware desde el reloj del sistema. En este caso también se utiliza el programa hwclock para setear el tiempo del reloj de hardware desde el reloj del sistema.

El programa hwclock puede utilizar diferentes maneras para obtener y setear los valores del RTC. El núcleo accede directamente a los registros de memoria del RTC a través de los puertos de E/S 0x70 y 0x71, hwclock puede actuar directamente sobre estos puertos. Pero la manera mas común de programar al RTC es hacer E/S sobre el archivo especial de dispositivo /dev/rtc, quien es manejado por el manejador del dispositivo rtc, aquí podemos separar entre dos familias de APIs RTC altamente compatibles. El manejador para las arquitecturas x86 con interface virtual /dev/rtc y el manejador genérico (el cual da soporte a una amplia variedad de chips RTC sobre todas las arquitecturas incluidos los sistemas con mas de un chip) con interface virtual /dev/rtc0, /dev/rtc1, /dev/rtcN, también posee otras interfaces pero la compatibilidad se da en estas interfaces.

Se utiliza un driver para programar al RTC, lo que se debe hacer es abrir el archivo especial del dispositivo para poder establecer la conexión entre el dispositivo RTC y un descriptor de archivo. Luego realizar la operación y posteriormente cerrar el descriptor del archivo para desconectar el dispositivo RTC con el descriptor de archivo. Las funciones de apertura y cierre se realizan con las llamadas a funciones estándar open y close. Para realizar la operación se utilizan los comandos ioctl [5] del dispositivo RTC, los cuáles son usados para configurar el dispositivo RTC. En el driver RTC, la función de lectura estándar read es usada para esperar por la interrupción del dispositivo RTC. Cuando se llama a la función de lectura, el proceso de usuario queda bloqueado hasta la generación de la próxima interrupción.

El RTC puede ser utilizado tanto como para ser un origen de reloj como para ser una fuente de eventos. En el primer caso se indica que puede ser utilizado para la interpolación del tiempo y en el segundo caso puede ser utilizado como posible fuente de interrupciones del sistema. Si bien el sistema operativo no lo utiliza para tales efectos lo tiene como posibilidad válida.

2.1.2. Time Stamp Counter (contador de sello de tiempo)

Desde la aparición de los procesadores Pentium [6] y algunos equivalentes (de AMD) se implantó un sistema de medición de tiempo llamado TSC, cuyo uso se generalizó rápidamente. TSC corresponde a las siglas de Time Stamp Counter. Es un registro de 64 bits que se encuentra en el núcleo del procesador y que va almacenando el número de ciclos de procesador acumulados desde el último inicio/reinicio del sistema. Tiene un refresco muy rápido, es preciso y accesible, de ahí que sea el más utilizado para el control del tiempo en la plataforma x86.

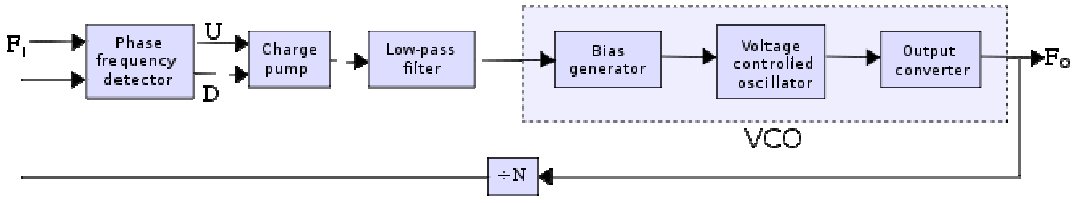
Desde el punto de vista a nivel físico todos los microprocesadores 80x86 incluyen un

pin de entrada CLK, que recibe la señal del reloj de un oscilador externo. Más específicamente, la señal que ingresa por este pin proviene de un generador de reloj de sistema; este dispositivo es el que contiene el oscilador de cristal de cuarzo (posee diferentes salidas tanto para el procesador como para otros dispositivos).
Se utiliza un chip sintetizador de frecuencia como generador de reloj porque construir un reloj de sistema de cristal de cuarzo (estable y preciso) arriba de 20Mhz es sumamente difícil y costoso. Por ejemplo el PCK2023 [7] es un sintetizador para un Pentium IV y otros procesadores similares. Posee salida de reloj de 33Mhz/48Mhz y más para PCI; tres diferentes señales de reloj para CPU (66, 100, 133 o 200Mhz) y su reloj de referencia es de 14.318Mhz (éste es el oscilador de cristal).

Notemos que la velocidad interna es la velocidad a la que funciona el microprocesador internamente. Como existen diferentes microarquitecturas dentro de un mismo microprocesador físico podemos encontrar los llamados cores/núcleos y dentro de éstos unidades lógicas de procesamiento. Éstos fueron introducidos con tecnologías como Hiper-Threading [8], tecnología con gran capacidad de procesamiento y rapidez que permite ejecutar múltiples hilos simultáneamente, y Multi-Core [9][10]. En realidad lo que nos interesa de estos diferentes niveles híbridos entre tener un sistema monoprocesador y uno multiprocesador es que se puede definir un tercer nivel de velocidad denominado velocidad de núcleo (de cada procesador lógico). El cual surge por el sistema de administración de energía. La unidad de control opera el reloj interno del microprocesador, y determina la razón de tiempo a la que el microprocesador opera; cada procesador lógico posee sus registros de control.

La señal de reloj para la CPU es la velocidad externa o de bus e indica la velocidad con la que se comunica el microprocesador y la placa base. En otros términos es la señal principal de la cual los demás dispositivos van a multiplicar/dividir para satisfacer sus necesidades de frecuencia de trabajo. Antiguamente se la referenciaba como FSB, el FSB [11] en los procesadores modernos ya no se utiliza dado que se ha reemplazado por QPI [12][13] o Hiper-Transport [14][15][16], éstas son diferentes tecnologías para comunicar el microprocesador con el chipset. La cifra por la que se multiplica la velocidad externa o de placa para dar la interna o del micro es el multiplicador; por ejemplo, un Pentium III a 450 MHz utiliza una velocidad de bus de 100 MHz y un multiplicador 4,5x.
La implementación real de un multiplicador (dispositivo que cambia la frecuencia de una señal) utiliza un circuito llamado 'phase-locked loop' (PLL o bucle enganchado en fase/bucle de enganche de fase/lazo de seguimiento de fase) [17][18][19]. Los diseños de PLL varían según su utilización, para PLL construidos dentro de un microprocesador, se pueden utilizar los osciladores anillo como osciladores controlados por voltaje (VCO). Ellos son construidos como un anillo de circuitos de retardos activos. Los osciladores de cristal son solo utilizados como la frecuencia de referencia de la PLL. [Figura 1].

Figura 1. Diseño Básico de PLL



También se pueden encontrar otras velocidades dentro del microprocesador; por ejemplo las Unidades Aritmético Lógicas (ALUs) corren con operaciones básicas con enteros al doble de frecuencia del procesador/ procesador lógico. Por lo tanto tenemos 4 velocidades de trabajo, la velocidad externa, la velocidad interna, la velocidad del núcleo y la velocidad de la ALU. El TSC, dependiendo de las características del microprocesador, corre a velocidad del núcleo o a velocidad interna.

Existen instrucciones en lenguaje ensamblador que nos permiten leer el registro Time Stamp Counter (TSC). Una de ellas es **rdtsc (**esta instrucción carga los 32 bits de la parte alta al registro EDX, y los 32 bits de la parte baja al registro EAX); también existe otra instrucción similar llamada **rdtscp** (obtiene información del TSC y también del microprocesador) [20][21]. Este registro es un contador que se incrementa con cada marca de CPU (1/CPU_HZ) y lo resetea a 0 cuando este es reseteado- si, por ejemplo, el reloj de la CPU marca (tick) a 1 GHz, el TSC se incrementa una vez cada un nanosegundo (10^9 veces por segundo). Aunque, hay que tener en cuenta que el intervalo de incremento varía dependiendo de las características del microprocesador, más adelante se explicará el problema que genera esta variación.

seconds = # cycles / frequency (velocidad de reloj del procesador en Hz)

A partir del Pentium Pro, los procesadores Intel han soportado la ejecución fuera de orden, donde las instrucciones no son necesariamente ejecutadas en el orden en que aparecen en el ejecutable. Esto puede causar que la instrucción **rdtsc** sea ejecutada más tarde de lo previsto, generando una cuenta de ciclos engañosa. Este problema puede ser resuelto por la ejecución de una instrucción de serialización. Tales como **cpuid** [22][23], para obligar a todas las instrucciones anteriores a terminar antes de permitir que el programa continúe (no es necesaria esta instrucción en los procesadores Pentium con tecnología MMX [24]) o utilizando directamente **rdtscp**. Otras variantes de serialización son **sfence**, **lfence** y **mfence** [25]. En la familia de procesadores 10h de AMD [26], se puede utilizar la instrucción **mfence** en lugar de **cpuid** como una instrucción de serialización [27][28]. El hecho de considerar al **rdtsc** como una instrucción serializada o no depende de la arquitectura utilizada; el precio a pagar por esto es una ligera lentitud en el gettimeofday() (entre 10 y 40 ciclos aproximadamente).
Cuando el microprocesador se encuentra en modo protegido/vitual 8086, una bandera llamada TSD (TIME STAMP DISABLE) en el registro de control CR4 permite al software controlar el nivel de privilegio en el que el registro TSC puede ser leído. Cuando TSD = 0, la instrucción **rdtsc** puede ser ejecutada en algún nivel de privilegio. Cuando TSD = 1, la instrucción solo puede ser ejecuta en el nivel de privilegio 0 (por lo tanto se deberá utilizar la interface provista por el sistema operativo para su utilización). Cuando se esté en modo de direccionamiento real, la instrucción **rdtsc** siempre está habilitada. El registro TSC también puede ser leído utilizando la instrucción **rdmsr**, siempre y cuando se este en el nivel de privilegio 0.

Linux y el TSC:

Linux toma ventaja del registro TSC para obtener mucho más precisas mediciones del tiempo que las emitidas por el Temporizador de Intervalos Programables. Para hacer esto, el sistema operativo determina la frecuencia con la que se incrementa el TSC mientras inicializa el sistema. Con la función calibrate_tsc calcula la frecuencia de contar el número de señales de reloj que ocurren en un intervalo de tiempo

relativamente largo. Esta constante de tiempo se produce por configurar adecuadamente uno de los canales del Temporizador de Intervalos Programable. El tiempo largo de ejecución de calibrate_tsc no crea problemas, ya que la función se invoca sólo durante la inicialización del sistema. Una vez hallada la frecuencia y posteriormente configurado el manejador del TSC, éste queda listo para su uso por parte del conjunto de funciones, variables y constantes del manejador genérico de orígenes de reloj, el cual maneja los posibles orígenes de reloj del sistema. Luego Linux puede utilizarlo para realizar la interpolación del tiempo.

2.1.3. Temporizador de Intervalos Programables (PIT)

El PIT es un dispositivo contador que emite una interrupción especial llamada interrupción temporizada (timer interrupt) cuando alcanza la cuenta programada, quien notifica al núcleo que un intervalo de tiempo ha transcurrido. El modo de operación del PIT puede ser one-shot o periódico. Los temporizadores one-shot interrumpen una sola vez, y después paran de contar. Los temporizadores periódicos interrumpen cada vez que alcanzan un valor específico (hacen reset automático).
Cada PC IBM-compatible incluye al menos un PIT, que suele ser implementado por un chip CMOS 8254 utilizando los puertos de E/S 0x40-0x43. El dispositivo usa un oscilador de cristal de 1.193182 MHz y contiene tres contadores de tiempo. El contador 0 es usado por el sistema operativo como contador de tiempo del sistema, el contador 1 es usado para el refresco de la memoria DRAM y el contador 2 para el altavoz de la computadora. Actualmente el PIT puede ser emulado por el Temporizador de Eventos de Alta Precisión (HPET).

Linux y el PIT:

En arquitecturas x86, Linux programa al PIT para emitir interrupciones temporizadas sobre el IRQ 0 en una frecuencia de (aproximadamente) 250 Hz, es decir, una vez cada 4 milisegundos. Aunque tengamos en cuenta que el intervalo de las marcas o la frecuencia de las interrupciones temporizadas depende de la arquitectura del hardware. Las máquinas más lentas tienen una marca de aproximadamente 10 milisegundos (100 interrupciones temporizadas por segundo). Los equipos más potentes tienen una marca de aproximadamente 1 milisegundo (1000 o 1024 interrupciones temporizadas por segundo). A este intervalo de tiempo se lo conoce como una marca (tick), y su duración en nanosegundos es almacenada en la variable tick_nsec. En una PC, tick_nsec puede ser automáticamente ajustada por el núcleo si el sistema se sincroniza con un reloj externo. El PIT puede ser utilizado tanto como para ser un origen de reloj como para ser una fuente de eventos.
Las marcas dan el tiempo para todas las actividades en el sistema. Aunque en la actualidad se está avanzando en el desarrollo de un kernel sin marcas. Por ejemplo la implementación parcial de las llamadas marcas dinámicas, que eliminan las marcas periódicas durante tiempos ociosos de CPU. Debido a que con una arquitectura de temporización flexible no es obligatorio interrumpir de forma periódica a la CPU sino hasta que realmente halla algo por hacer. Por ello, según esta implementación parcial, cuando la CPU entra a su estado ocioso, éste verifica el próximo evento pendiente de temporizador. Si el próximo evento se produce después de la próxima marca periódica, la marca periódica es apagada; en su lugar, el temporizador es programado para interrumpir cuando el próximo evento llegue. La CPU puede descansar hasta el próximo evento, amenos que llegue una interrupción primero. Una vez que el procesador sale del

estado ocioso, se reestablece la marca periódica.
Eliminar las marcas en tiempo ocioso de CPU es bueno para empezar, pero aun resta llevar el mismo concepto a tiempo donde la CPU no esta ociosa, para completar realmente una implementación completa de un sistema con marcas dinámicas. La implementación parcial de marcas dinámicas deja espacio para lograr un sistema completo sin marcas, donde el time slice o quantum (periodo de tiempo en el cual se permite correr a un proceso) sea controlada por el scheduler (planificador). También, para disponer de perfiles de frecuencia variable, y una eliminación completa de jiffies (variable que indica las marcas que han pasado desde el inicio/reinicio del sistema) en el futuro.

En términos generales, las marcas mas cortas resultan en temporizadores de mayor resolución y un tiempo de respuesta más rápido cuando se realiza multiplexación de E/S sincrónica. La multiplexación es cuando se utilizan dos canales de información en un solo medio de transmisión. Sin embargo, marcas más cortas requieren que la CPU gaste una fracción mayor de su tiempo en modo kernel, es decir, una menor fracción de tiempo en modo usuario. Como consecuencia de ello, los programas de usuario corren más lento.

Unos pocos macros en el código Linux dan algunas constantes que determinan la frecuencia de las interrupciones temporizadas.

HZ da el número de interrupciones temporizadas por segundo, es decir, su frecuencia. Este valor se fija en 250 para PCs IBM.
PIT_TICK_RATE da el valor 1.193.182, que es la frecuencia del oscilador interno del chip i8253/i8254.
LATCH da la relación entre PIT_TICK_RATE y HZ, redondeada al entero más cercano. Se utiliza para programar el PIT.

2.1.4. Temporizador del Controlador de Interrupciones Programable Avanzado Local (LAPIC)

Antes de explicar el temporizador del LAPIC vamos a explicar que es un PIC, IO/APIC y finalmente el LAPIC dentro del cual se encuentra el temporizador.

2.1.4.1. Controlador de Interrupciones Programable (PIC)

El PIC es un dispositivo usado para combinar varias fuentes de interrupciones sobre una o más líneas de CPU, mientras que permite que niveles de prioridad sean asignados a sus entradas de interrupción. Los modos de prioridad comunes de un PIC incluyen prioridades estáticas, rotativas y en cascada.

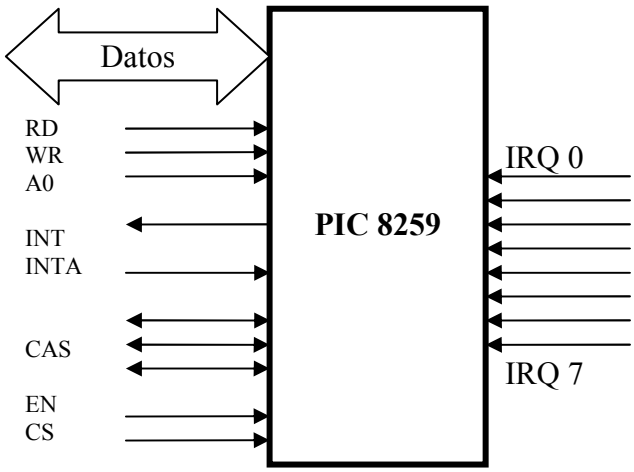
A continuación se describirán las señales de la estructura externa del PIC [Figura 2]:

- IRQ0-IRQ7: Peticiones de interrupción de los periféricos, de mayor a menor prioridad.
- INT: Petición de interrupción al procesador.
- INTA (Interrupt Acknowledgement): Reconocimiento/aceptación de la interrupción por parte del procesador.
- CS (chip select): Para habilitar a la CPU a leer o escribir en los registros del PIC.

Es utilizada para programar al PIC.

- CAS2-CAS0: Líneas para la conexión en cascada de varios PIC 8259. Actúan como salida del PIC maestro y como entrada de los PIC esclavos.
- EN: Indica si el PIC actúa como maestro o como esclavo cuando hay varios PICs encadenados en cascada.
- RD, WR (read, write): Permiten leer o escribir en los registros de control del PIC 8259.
- A0: Única línea del bus de direcciones usada para seleccionar los registro de control.
- Bus de datos Bidireccional: Intercambio de datos ente el PIC y el resto de los componentes de la computadora (memoria y procesador).

Figura 2: Esquema de la estructura externa de un PIC tradicional estilo 8259A.



Cuando es necesario atender a las peticiones de interrupción de más de ocho periféricos se pueden conectar varios PICs en cascada. Se utilizan dos niveles de controladores: en el primero tenemos al PIC maestro, y en el segundo a los esclavos. Solo la línea de salida INT del maestro esta conectada a la línea de entrada INTR del procesador. Los dispositivos disponen de una línea de salida que les permite emitir solicitudes de interrupción para que el procesador deje de hacer lo que estaba haciendo y atienda al dispositivo. Ésta línea es designada como una solicitud de interrupción (IRQ).

Dentro de la estructura interna, los PICs típicamente tienen un conjunto común de registros de 8 bits:

- Interrupt Request Register (IRR), o registro de solicitudes de interrupción: El IRR especifica qué interrupciones están pendientes de reconocimiento, y es típicamente un registro interno que no puede ser accedido directamente, cambia un bit a 1 en la posición de la interrupción solicitada.
- In-Service Register (ISR), o registro de interrupciones en servicio: El registro ISR específica qué interrupciones han sido reconocidas, pero todavía están esperando por un final de interrupción (EOI).
- Interrupt Mask Register (IMR), o registro de mascara/enmascaramiento de interrupciones: El IMR especifica qué interrupciones deben ser ignoradas y no ser reconocidas. Un esquema simple de registros como éste, permite que estén pendientes en un mismo tiempo hasta dos distintas peticiones de

interrupción, una esperando por reconocimiento, y una esperando por EOI. Éste es el único registro al que se puede acceder a través del puerto 21h para el chip maestro y a través del puerto A1h para el chip esclavo.

También dentro de la estructura interna del PIC nos encontramos con:

- Lógica de gestión de prioridad: determina qué interrupción, de las solicitadas en el IRR, debe ser atendida primero.
- Buffer del bus de datos: conecta al PIC con el bus de datos de la placa principal de la PC.
- Lógica de lectura y escritura: acepta los comandos que envía la CPU; transfiere el estado del PIC hacia el bus de datos.
- Buffer de cascada/comparador: almacena y compara las identificaciones de todos los PICs del sistema.

Las líneas IRQ son secuencialmente numeradas a partir de 0, por lo que la primera línea de IRQ es generalmente nombrada como IRQ0. En Intel la asociación de vectores o números de interrupción se indica con el mapeo IRQn con n+32. El mapeo entre IRQs y los vectores pueden ser modificados mediante la emisión adecuada de instrucciones de E/S hacia los puertos del Controlador de Interrupciones Programable.

Finalmente, podemos decir que el PIC realiza las siguientes acciones:

- **1**. Supervisa las líneas IRQ, verificando por señales producidas.
- **2**. Si se produce una señal sobre una línea IRQ:
  - **A.** Convierte la señal recibida en el correspondiente vector (dicho así por Intel al numero de interrupción).
  - **B.** Almacena el vector en un puerto de E/S del Controlador de Interrupción Programable, lo que permite a la CPU leer el vector a través del bus de datos.
  - **C.** Envía una señal levantada hacia el pin INTR del procesador; es decir, emitir una interrupción.
  - **D.** Espera hasta que la CPU reconozca la señal de interrupción por escribir dentro de uno de los puertos de E/S de los PICs ; es decir, la CPU activa la línea de salida INTA para reconocer la interrupción; cuando esto ocurra, borra la línea INTR.
- **3**. Ir al paso **1**.

2.1.4.2. Controlador de Interrupciones Programable Avanzado Local (LAPIC)

La descripción anterior se refiere a PICs diseñados para sistemas monoprocesador. Si el sistema incluye una única CPU, la línea de salida del PIC maestro se puede conectar de una manera sencilla al pin INTR de la CPU. Sin embargo, si el sistema incluye dos o más CPUs, tenemos un problema.

El LAPIC es un controlador de interrupciones programable avanzado incorporado en la CPU (también denominado APIC local) y diseñado por y para el multiproceso, concretamente para poder incorporar múltiples microprocesadores a la placa madre. Al hablar del LAPIC también tenemos que tener en cuenta al I/O APIC de la placa madre que consiste en una mejora del clásico PIC y la ventaja es que ofrece más líneas de

IRQs y un manejo mas rápido de las mismas.

Como están fuertemente relacionados los explicaremos juntos para que se entienda mejor su relación y comportamiento.

I/O APIC y LAPIC:

Ser capaz de entregar interrupciones a cada CPU en el sistema es crucial para explotar totalmente el paralelismo de la arquitectura SMP, en la que los microprocesadores comparten el acceso a memoria. Por esta razón, Intel ha introducido un componente designado como el Controlador de Interrupciones Programable Avanzado de E/S (I/O APIC), que sustituye al antiguo PIC. Contiene una tabla de redirección, que se usa para enrutar las interrupciones que recibe por parte de los periféricos hacia una o más LAPICs, esta unidad es la que está directamente relacionada con la entrada/salida.

Las características principales del I/O APIC son las siguientes:

- Un conjunto de 24 líneas de IRQ.
- Una Tabla de Redirección de interrupción de 24-entradas.
- Registros programables.
- Un canal de comunicación para enviar y recibir mensajes entre el I/O APIC y los LAPICs, llamado bus APIC o bus inter-APIC. Antiguamente el bus APIC era implementado por medio de un canal de comunicación particular, en la actualidad el bus APIC es implementado por medio del bus del sistema.
- Puede trabajar en un entorno monoprocesador y multiprocesador.
- Gestiona interrupciones en un entorno multiprocesador distribuyendo las interrupciones entre los procesadores simétricamente, bien de forma estática o bien de forma dinámica.
- Tiene una latencia de atención a las interrupciones menor debido a la eliminación de los ciclos de reconocimiento de interrupción. Por el uso del canal de comunicación entre el I/O APIC y los LAPICs, el proceso de reconocimiento de una interrupción no se realiza al modo estándar como en el PIC.

Por otra parte, todas las CPUs actuales de Intel incluyen un LAPIC dentro del microprocesador, el cual gestiona todas las interrupciones externas para el microprocesador del que forma parte. Además, es capaz de aceptar y generar interrupciones interprocesador entre las LAPICs. Puede soportar hasta vectores de 224 IRQ de una I/O APIC. Los números entre el 0 y el 31 están reservados para el manejo de excepciones de los microprocesadores x86.

Las características principales del LAPIC son las siguientes:

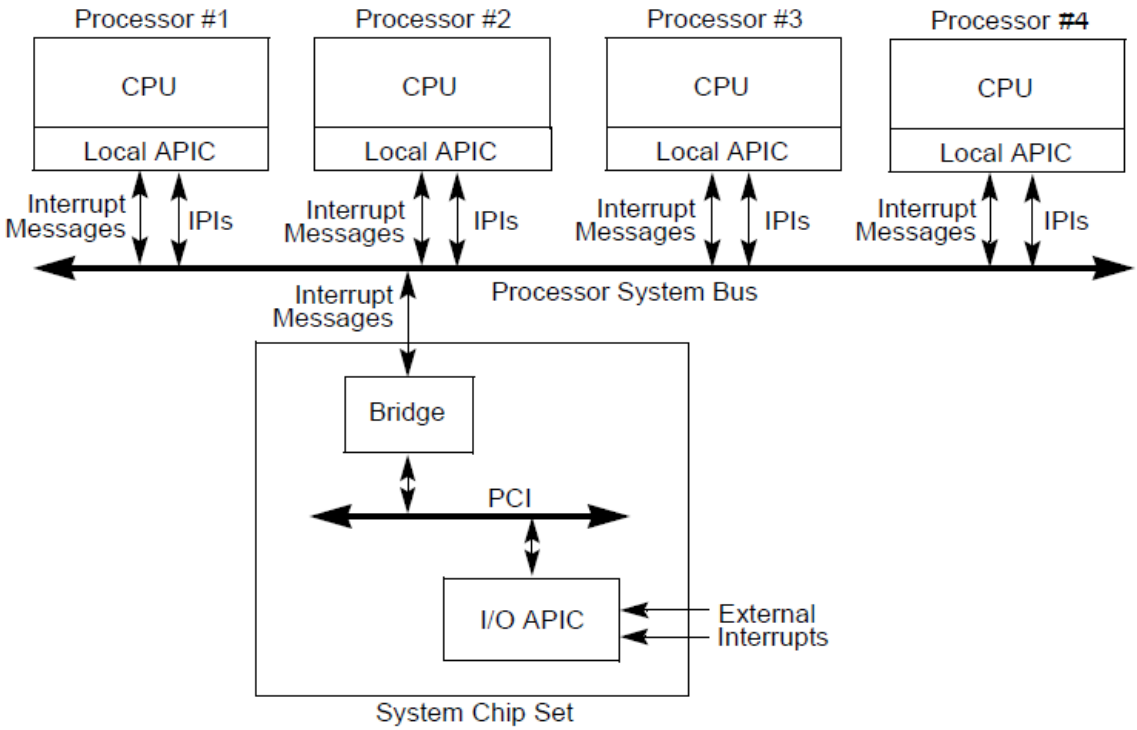
- Registros de 32 bits de longitud.
- Gestiona la recepción de interrupciones a través del canal de comunicación que se da entre éste y los demás LAPICs e I/O APIC.
- Maneja interrupciones pendientes, anidamiento de interrupciones, enmascaramiento.
- Recibe nuevos mensajes del canal de comunicación entre LAPICs e I/O APIC y gestiona su atención emitiendo la interrupción hacia el procesador si

la nueva interrupción es de mayor prioridad que la que está siendo atendida. O reteniéndola si es de menor prioridad o ha sido particularmente enmascarada.

- Se encarga de la gestión de las interrupciones al modo estándar utilizando el así llamado protocolo INTR/INTA/EOI. Esto es, generando o invocando en el microprocesador un ciclo especial de bus de reconocimiento de interrupciones cuando recibe una señal por la entrada INTR.
- También atiende y realiza interrupciones interprocesador (para entablar comunicación directa entre los microprocesadores). A través de dos líneas IRQ adicionales LINT0 y LINT1 reservados para las interrupciones locales.
- Por último incorpora un temporizador.

Todos los LAPICs están conectados a un I/O APIC externo, dando lugar a un sistema multi-APIC.
Un bus APIC conecta al I/O APIC hacia los LAPICs [Figura 3]. Las líneas IRQ procedentes de los dispositivos están conectadas al I/O APIC, que actúa como un enrutador con respecto a los LAPICs. En las placas madre de las Pentium III y microprocesadores anteriores, el bus APIC fue un bus serial de 3 líneas, comenzando con el Pentium 4, el bus APIC es implementado por medio del bus del sistema.

Figura 3. Sistema Multi-APIC que se comunica a través del bus del sistema



A diferencia de los pins IRQ de la 8259A, la prioridad de la interrupción no está relacionada con el número de pin. Cada entrada en la Tabla de Redirección puede ser programada individualmente para indicar el vector y la prioridad de interrupción, el microprocesador destino, y cómo el microprocesador es seleccionado. La información en la Tabla de Redirección es utilizada para traducir cada señal externa IRQ en un mensaje para una o más unidades LAPICs a través del canal de comunicación entre el I/O APIC y los LAPICs.

Las solicitudes de interrupción procedentes de dispositivos de hardware externos pueden distribuirse entre las CPUs disponibles en dos formas:

- Distribución Estática: La señal IRQ se entrega a las LAPICs que figuran en la correspondiente entrada de la Tabla de Redirección. La interrupción se entrega a una determinada CPU, a un subconjunto de CPUs, o a todas las CPUs a la vez (modo broadcast).
- Distribución Dinámica: La señal IRQ se entrega a la LAPIC del procesador que está ejecutando el proceso con la prioridad más baja.

En cuanto a la distribución dinámica, algunas LAPICs tienen un Registro de Prioridad de Tarea programable (TPR), que se utiliza para computar la prioridad del proceso que se está ejecutando actualmente. Intel espera que este registro sea modificado por el núcleo del sistema operativo por cada cambio de proceso. Si dos o más CPUs comparten la prioridad más baja, la carga se distribuye entre ellos mediante una técnica llamada arbitraje. A cada CPU se le asigna una prioridad de arbitraje que va desde 0 (más baja) a 15 (más alta) en el registro de prioridad de arbitraje del LAPIC; cada LAPIC cuenta con un valor único. Cada vez que una interrupción se entrega a una CPU, su correspondiente prioridad de arbitraje automáticamente se pone a 0, mientras que las prioridades de arbitraje de todas las demás CPU se incrementan. Cuando el registro de prioridad de arbitraje se vuelve más grande que 15, se configura con la prioridad de arbitraje previa de la CPU que ha ganado incrementada en 1. Por tanto, las interrupciones se distribuyen en una forma round-robin entre las CPUs con la misma prioridad de tarea. Tengamos en cuenta que la LAPIC del Pentium 4 no tiene un registro de prioridad de arbitraje; el mecanismo de arbitraje está en el circuito de arbitraje del bus. Los manuales de estado Intel establecen que si el sistema operativo periódicamente no actualiza los registros de prioridad de tareas, la performance podría ser inferior al óptimo porque las interrupciones podrían ser siempre atendidas por la misma CPU.

Además de la distribución de las interrupciones entre los procesadores, los sistemas multi-APIC permiten a las CPUs generar interrupciones interprocesador. Cuando una CPU desea enviar una interrupción a otra CPU, almacena el vector de interrupción y el identificador del LAPIC del objetivo (target) en el Registro de Comando de Interrupción (ICR) de su propia LAPIC. Posteriormente, un mensaje se envía a través del canal de comunicación entre LAPICs hacia el LAPIC del objetivo, quien emite la correspondiente interrupción a su propia CPU.

La mayoría de los actuales sistemas monoprocesador incluyen un chip I/O APIC, que puede ser configurado de dos formas distintas:

- Como un estándar PIC externo estilo 8259A conectado a la CPU. El LAPIC es deshabilitado y las dos líneas IRQ local LINT0 y LINT1 son configuradas, respectivamente, como los pins INTR y NMI.
- Como un estándar externo I/O APIC. El LAPIC es habilitado y todas las interrupciones externas se reciben a través del I/O APIC.

2.1.4.3. Detalles del temporizador del LAPIC:

El temporizador es un dispositivo que puede emitir interrupciones periódicas o en un

disparo (one-shot), que es similar al PIT. Sin embargo, hay unas pocas diferencias:

- El contador del temporizador del LAPIC es de 32-bits de longitud, mientras que el contador del temporizador del PIT es de 16-bits de longitud. De esta forma los temporizadores locales pueden programarse para emitir interrupciones a muy baja frecuencias; el contador almacena el número de marcas que deben transcurrir antes de que la interrupción sea emitida.
- El temporizador del LAPIC esta físicamente enlazado a la CPU en cambio el PIT esta en un circuito separado.
- El temporizador del LAPIC envía una interrupción sólo para su procesador, mientras que el PIT levanta una interrupción global, que puede ser manejada por cualquier CPU en el sistema.
- El temporizador del LAPIC se basa sobre la señal del canal de comunicación entre el I/O APIC y los LAPICs. Puede ser programada de tal manera para decrementar el contador del temporizador cada 1, 2, 4, 8, 16, 32, 64, o 128 señales del bus del reloj del sistema. Por el contrario, el PIT tiene su propio oscilador de reloj interno.

La resolución mínima del temporizador del LAPIC es en la magnitud de los microsegundos. Como la velocidad de las modernas PCs basadas en x86 comienzan en 100MHz, la resolución minima es de 0,01 microsegundos o 10 nanosegundos. Pero debido al tiempo de cálculo necesario para realizar el cambio a la rutina de servicio de interrupción (salvar la información de contexto entre otras cosas) la medida llega a 1 microsegundo. Consecuentemente esto es de 1000 a 10000 veces más alta la precisión que el temporizador por defecto de la PC; depende de la frecuencia a la que esté el núcleo.

Modos del temporizador:

Modo periódico: Para el modo periódico, el software setea un valor de cuenta inicial y el LAPIC usa éste como el contador actual. El LAPIC decrementa el contador hasta que se vuelva cero, entonces genera una interrupción y resetea el contador con el valor inicial, posteriormente comienza a decrementar el contador de nuevo. De esta forma el LAPIC genera interrupciones a una tasa fija dependiendo del valor inicial. El contador es decrementado a una tasa que depende de la frecuencia externa de la CPU (frecuencia de bus) dividida por el valor del registro “Divide Configuration Register” del LAPIC. Por ejemplo, para una CPU de 2.4 GHz con una frecuencia externa/bus de 800 MHz. Si el registro de configuración de división es seteado a “divide por 4” y la cuenta inicial es seteada a 123456; entonces el temporizador del LAPIC decrementará el contador a una tasa de 200 MHz. Por lo que se generará una interrupción cada 617.28 us (1/tasa de IRQs en HZ), dando una tasa de IRQs de 1620.01 Hz (tasa de decremento en HZ/valor inicial del contador).

Modo One-Shot (un disparo): Funciona de la misma manera que el modo periódico; salvo que este no resetea el valor actual del contador al valor inicial de la cuenta cuando el valor actual se vuelve cero. De esta forma, el software tiene que setear el nuevo valor cada vez que quiera mas interrupciones temporizadas.

También existe un tercer modo, ésta es una extensión que solo es soportada en las últimas CPUs. Modo TSC-Deadline (limite TSC): Este modo es muy diferente a los

otros dos modos. En lugar de usar la frecuencia externa de la CPU para decrementar la cuenta el software setea un valor limite y el LAPIC genera una interrupción temporizada cuando el valor del TSC es mas grande o igual al valor limite. El temporizador es implementado vía un nuevo registro de 64 bits, IA32_TSC_DEADLINE_MSR. A pesar de estas diferencias, el software puede usar éste de la misma manera que el modo one-shot. La ventaja (comparado con el modo one-shot) es que se obtiene una alta precisión (porque el TSC de la CPU corre a la frecuencia interna de la CPU la cual es mucho mayor que la frecuencia externa).

Linux y el Temporizador del LAPIC:

Linux provee soporte para los modos de funcionamiento periódico y one-shot por default y existen parches para el soporte al modo TSC-Deadline. Los modos de funcionamiento permiten al LAPIC ser utilizado como fuente de eventos en el sistema.

Linux posee una tabla de vectores de interrupciones el cual reserva algunos vectores para dispositivos específicos, mientras que el resto pueden ser manejados dinámicamente. Uno de estos vectores reservados corresponde al temporizador del LAPIC (239), por lo tanto su manejador está definido estáticamente. Tanto en monoprocesador o en multiprocesador se setea el puerto de interrupción del IDT correspondiente al vector 239 con el manejador de interrupción de bajo nivel apic_timer_interrupt. El procesador usa al IDT (Tabla de Descripción de Interrupción) para determinar la correcta respuesta a interrupciones y excepciones En sistemas monoprocesador el núcleo no permite setearle otro manejador; en sistemas multiprocesador no se permite su uso porque es utilizado por el núcleo para llevar a cabo parte de las actividades relacionadas al manejo del tiempo.

En general podríamos decir que las actividades para el manejo del tiempo las podemos dividir en dos partes, unas en aquellas que pueden ser realizadas por cualquier CPU y otras que dependen de la CPU sobre la cual se ejecutan. Aquí es donde se utiliza el manejador del temporizador del LAPIC para ejecutar las actividades que dependen de la CPU, más adelante se explicará cuáles son estas actividades particulares.

2.1.5. Temporizador de eventos de alta precisión (HPET)

El temporizador de eventos de alta precisión (HPET, High Precisión Event Timers) es el último chip temporizador, desarrollado conjuntamente por Intel y Microsoft, el mismo no es una característica del procesador sino del chipset de la placa madre. Fue desarrollado para solucionar los requerimientos de tiempos de aplicaciones multimedia y otras aplicaciones sensitivas al tiempo (pj: sincronización de streams de audio y video digital; planificación de threads, tareas o procesos). Originalmente fue llamado Temporizador Multimedia. Además de extender las capacidades y precisión del sistema, el HPET mejora el rendimiento del sistema.
El HPET proporciona un número de temporizadores de hardware que pueden ser explotados por el núcleo. Básicamente, el chip incluye hasta ocho contadores independientes. El HPET tiene una granularidad programable, en microsegundos. Adicionalmente, el temporizador tiene una precisión de nanosegundos de manera que las interrupciones no sean generadas o entregadas tardíamente, y de esta forma se minimiza la desviación por lo que es consistentemente fiable.

Características principales:

Incremento monotónico del contador: Cada contador se incrementa monotónicamente y se ve impulsado por su propia señal de reloj, cuya frecuencia es de al menos 10 MHz, por lo que el contador se incrementa por lo menos una vez en 100 nanosegundos. Cualquier contador está asociado con a lo sumo 32 temporizadores, cada uno de los cuáles está compuesto por un comparator (comparador) y un registro de match conformando un bloque temporizador. [Figura: 4]. En total se pueden soportar 256 temporizadores.

Comparadores independientes: El comparador es un circuito que comprueba el valor en el contador contra el valor en el registro de match. Emite una interrupción de hardware independiente si se encuentra una igualdad, de esta forma cada comparador puede ser utilizado independientemente.

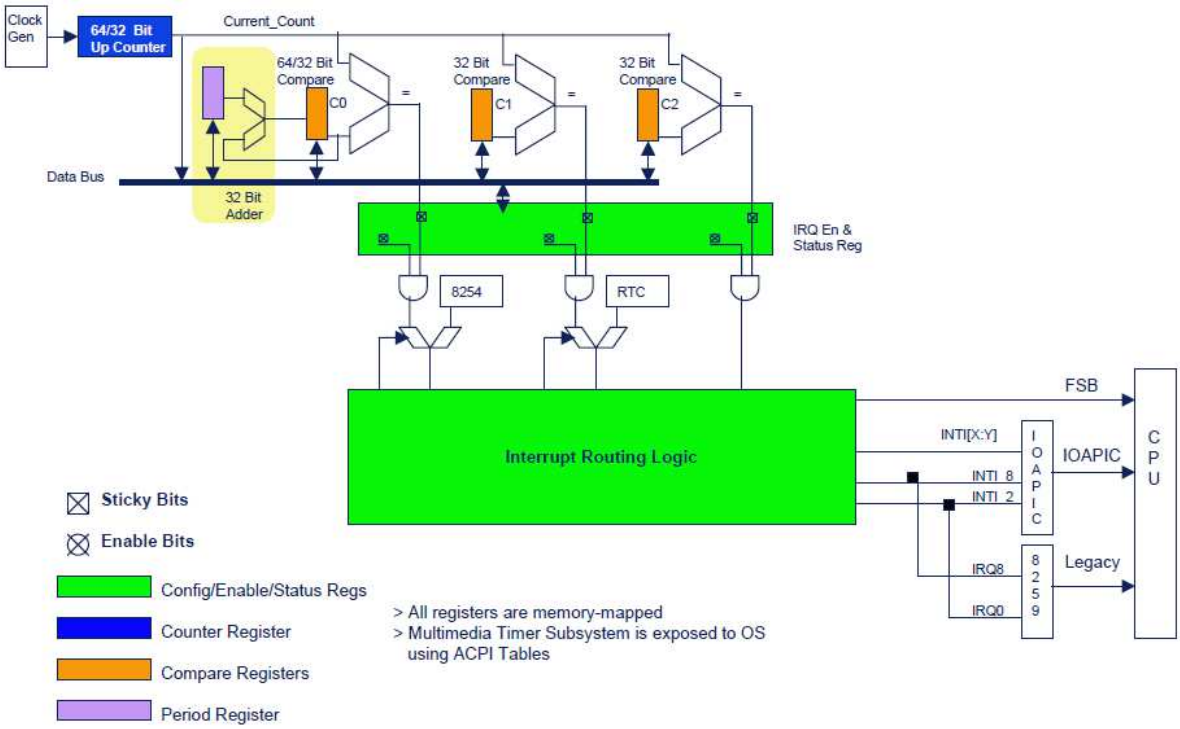
Interrupciones exclusivas: además de generar interrupciones independientes, cada comparador tiene una única (no compartida) interrupción exclusiva en modo de operación de ruteo de interrupción no compartido. Esta opción se debe a que el procesamiento de una rutina de servicio de interrupción (ISR) de una interrupción compartida es más cara que el procesamiento de una interrupción exclusiva. Ej: mediante los registros de configuración se puede hacer que las interrupciones del temporizador 0 sean enrutadas hacia el IRQ 0, el temporizador 1 sea enrutado al IRQ 8, y asi.

Soporte modo periódico y one-shot: Cada comparador soporta funcionalidad one-shot. Modo en el que solo se genera una única interrupción en un tiempo especifico, si subsecuentes interrupciones son requeridas después de que una interrupción sea generada, el temporizador debe ser reprogramado. Y al menos un comparador soporta funcionalidad periódica (genera interrupciones de manera regular).

Mapeo en memoria de registros: Los registros del temporizador están definidos como un conjunto no indexado y mapeados a memoria de manera que cada temporizador puede ser accedido directamente sin una búsqueda.

Ancho de registros de 32-bits o 64-bits: El registro del comparador y el registro del contador principal pueden ser de 32 o 64-bits de ancho.

Figura 4.Diagrama de bloques de la arquitectura HPET.



Linux y el HPET:

El chip HPET puede programarse a través de sus registros asociados a los temporizadores, mapeados dentro del espacio de memoria. El BIOS establece el mapeo durante la fase de arranque y reporta al núcleo del sistema operativo su dirección inicial de memoria. Los registros del HPET permiten al núcleo leer y escribir los valores de los contadores y de los registros de match, para programar interrupciones en un disparo, y para habilitar o deshabilitar las interrupciones periódicas en los temporizadores que los soportan [29][30]. El HPET provee una emulación para el PIT a través del temporizador 0. También, posee una emulación parcial para el RTC, debido a que no todas las implementaciones de la especificación del HPET proveen un modo periódico para el temporizador 1, el cual es un canal necesario para emular el RTC. El HPET en modo de emulación mapea a las interrupciones asociadas al PIT o RTC; dando funcionalidad de interrupción del PIT/RTC en software. La entrega de interrupción cuando se emula al PIT/RTC se realiza vía el I/O APIC, opcionalmente puede ser realizada por otras vías (por ejemplo vía FSB, Front Side Bus [11]). Podemos resumir que las interrupciones asociadas con los temporizadores tienen varias opciones de mapeo de interrupciones. La primer forma de ruteo es: El temporizador 0, es enrutado al IRQ 0 vía no I/O APIC o IRQ2 vía I/O APIC. El temporizador 1 es enrutado al IRQ 8 tanto en vía no I/O APIC como en vía I/O APIC. El resto de los temporizadores son enrutados según sus bits de ruteo particular de la configuración de sus registros. La segunda forma es: Cada temporizadores tiene su propio control de ruteo. Las interrupciones pueden ser enrutadas a varias interrupciones en el I/O APIC. La tercer forma es: Las interrupciones son mapeadas directamente a través de interrupciones FSB.
Debido a que el HPET puede ser configurado, el mismo puede ser utilizado tanto como para ser un origen de reloj como para ser una fuente de eventos.

2.1.6. Temporizador de Administración de Energía ACPI

Antes de entrar en detalle del temporizador expliquemos que es el ACPI. El ACPI es la Interfaz Avanzada de Configuración y Energía (Advanced Configuration and Power Interface). Es un estándar resultado de la actualización del APM (Advanced Power Management, es una API que permite al BIOS administrar la energía del sistema) a nivel de hardware. Controla el funcionamiento del BIOS y proporciona mecanismos avanzados de gestión y ahorro de energía. Lo que busca es la independencia del BIOS con respecto al sistema operativo.
El Temporizador de Administración de Energía de la Interfaz Avanzada de Configuración y Energía (ACPI PMT, Advanced Configuration and Power Interface Power Management Timer) es otro dispositivo de reloj incluido en casi todas las placa madre basadas en ACPI. Su señal de reloj tiene una frecuencia fija de alrededor de 3,58 MHz. El dispositivo es en realidad un simple contador incrementado con cada marca de reloj, el tamaño del registro es de 24 o 32-bits; para leer el valor actual del contador, el núcleo accede a un puerto de E/S, cuya dirección está determinada por la BIOS durante la fase de inicialización.

Linux y el APIC PMT:

Linux permite que el APIC PMT pueda ser utilizado como posible origen de reloj.

2.2. Conclusión

Para finalizar este capítulo podemos decir que los dispositivos de hardware anteriores pueden ser utilizados algunos como posibles orígenes de reloj y otros como posibles fuentes de eventos. El sistema operativo provee un manejador para cada uno de estos dispositivos el cual dependiendo de sus posibilidades implementa las interfaces necesarias para su uso por el conjunto de funciones, variables y constantes tanto del clocksource como del clockevent, los cuáles se explicaran en capítulos posteriores.

3. ESTADO DEL ARTE: Sincronización del Tiempo en Sistemas Distribuidos

En un sistema distribuido los procesos no solo se comunican sino que también cooperan y se sincronizan. En dichos sistemas los métodos de sincronización de los sistemas con una CPU no son adecuados (en general se basan en la existencia de memoria compartida). Por lo tanto hay necesidad de disponer de otros modelos de sincronización entre procesos para un sistema distribuido. Existen modelos que utilizan al tiempo como elemento fundamental para realizar la sincronización. Estos modelos de sincronización entre procesos disponen de algoritmos que requieren que a su vez el tiempo este sincronizado para un correcto funcionamiento. Por ello es que en los algoritmos distribuidos que dependen del tiempo hay una fuerte necesidad de sincronización del mismo. Generalmente los algoritmos distribuidos tienen las siguientes propiedades [1]:

- La información relevante se distribuye entre varias máquinas.
- Los procesos toman las decisiones solo con base en la información disponible en forma local.
- Debe evitarse un único punto de fallo en el sistema.
- No existe un reloj común o alguna otra fuente precisa del tiempo global.

Los primeros tres puntos indican que es inaceptable reunir toda la información en un solo lugar para su procesamiento, pero lograr la sincronización sin centralización requiere hacer las cosas distintas al caso de los sistemas operativos tradicionales. El último punto también es crucial:

- En un sistema centralizado el tiempo no es ambiguo (en sistemas monoprocesador). Si el proceso A pide la hora y un poco después, el proceso B también la pide, el valor de B es mayor o igual al valor de A.
- En un sistema distribuido no es trivial poner de acuerdo a todas las máquinas en la hora.
- Se requiere un acuerdo global en el tiempo, la falta de sincronización en los relojes puede ser drástica en procesos dependientes del tiempo.

Dado que la sincronización del tiempo es fundamental en los algoritmos distribuidos que lo utilizan, a continuación analizaremos los siguientes tipos de relojes en conjunto con los algoritmos utilizados para sincronizarlos:

- Relojes Lógicos:
  - Algoritmo de Lamport.

- Relojes Físicos:
  - Algoritmo de Cristian.
  - Algoritmo Berkeley.
  - Algoritmo de Promediación.

3.1. Relojes Lógicos

Las computadoras poseen un circuito para el registro del tiempo conocido como dispositivo reloj. Es un cronómetro consistente en un cristal de cuarzo de precisión sometido a una tensión eléctrica que:

- Oscila con una frecuencia bien definida que depende de:
  - La forma en que se corte el cristal.
  - El tipo de cristal.
  - La magnitud de la tensión.
- A cada cristal se le asocian dos registros:
  - Registro contador.
  - Registro mantenedor.
- Cada oscilación del cristal decrementa en "1" al contador.
- Cuando el contador llega a "0":
  - Se genera una interrupción.
  - El contador se vuelve a cargar mediante el registro mantenedor.
- Se puede programar un cronómetro para que genere una interrupción "x" veces por segundo.
- Cada interrupción se denomina marca de reloj.

Para una computadora monoprocesador y un reloj, no importan los pequeños desfasajes del reloj puesto que todos los procesos de la maquina usan el mismo reloj y tendrán consistencia interna. Lo que importa son los tiempos relativos. Para varias computadoras con sus respectivos relojes es imposible garantizar que los cristales de computadoras distintas oscilen con la misma frecuencia, lo que provocará una pérdida de sincronía en los relojes (de software). Es decir que tendrán valores distintos al ser leídos. La diferencia entre los valores del tiempo se llama distorsión del reloj y podría generar fallas en los programas dependientes del tiempo.

Lamport demostró que la sincronización de relojes es posible y presentó un algoritmo para lograrlo, también señaló que la sincronización de relojes no tiene que ser absoluta:

- Si 2 procesos no interactúan no es necesario que sus relojes estén sincronizados.
- Generalmente lo importante no es que los procesos estén de acuerdo en la hora, pero sí importa que coincidan en el orden en que ocurren los eventos.

Para estos algoritmos lo que importa es la consistencia interna de los relojes:

- No interesa su cercanía particular al tiempo real (oficial).
- Los relojes se denominan relojes lógicos.

En cambio los relojes físicos son relojes que:

- Deben ser iguales (estar sincronizados).
- No deben desviarse del tiempo real más allá de cierta magnitud.

3.1.1. Algoritmo de Lamport

Para sincronizar los relojes lógicos, **Lamport** definió la relación "ocurre antes de". La expresión "a -> b" se lee "a ocurre antes de b" e indica que todos los procesos coinciden en que primero ocurre el evento "a" y después el evento "b". La relación se puede observar de manera directa en dos situaciones:

- Si "a" y "b" son eventos en el mismo proceso y "a" ocurre antes de "b", entonces "a –> b" es verdadero.
- Si "a" es el evento del envío de un mensaje por un proceso y "b" es el evento de la recepción del mensaje por otro proceso, entonces "a -> b" también es verdadero. Un mensaje no se puede recibir antes de ser enviado o al mismo tiempo en que se envía, puesto que tarda en llegar una cantidad finita de tiempo.

"Ocurre antes de" es una relación transitiva, de modo que si "a –> b" y "b –> c", entonces "a –> c". Si dos eventos, "x" e "y", están en procesos diferentes que no intercambian mensajes, entonces "x –> y" no es verdadero, pero tampoco lo es "y –> x". Se dice que estos eventos son eventos concurrentes, lo que significa que nada se puede decir acerca del momento en el que ocurren o cual de ellos es el primero.

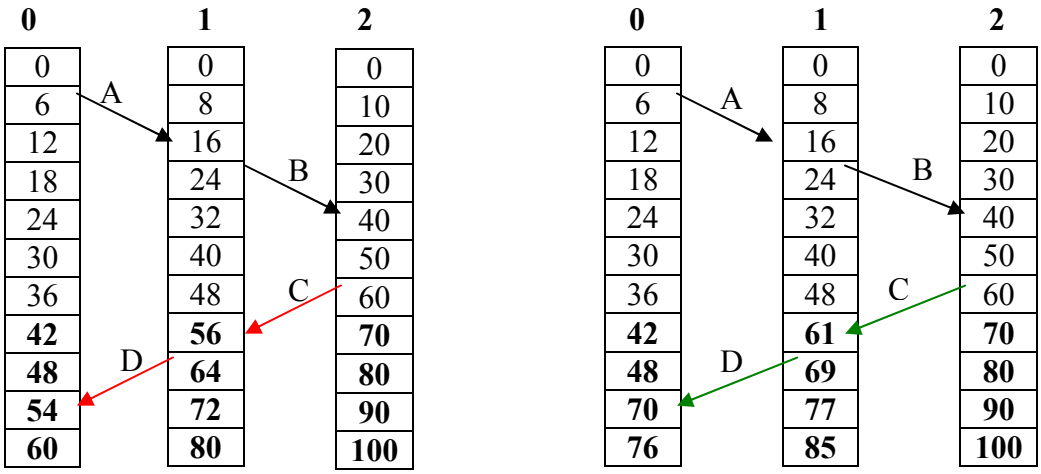
El algoritmo de Lamport asigna tiempos a los eventos. Necesitamos una forma de medir el tiempo tal que a cada evento "a", le podamos asociar un valor del tiempo "C(a)" en el que todos los procesos estén de acuerdo, se debe cumplir que:

- Si "a –> b" entonces "C(a) < C(b)".
- El tiempo del reloj, "C", siempre debe ir hacia adelante (creciente), y nunca hacia atrás (decreciente). Se pueden hacer correcciones al tiempo al sumar un valor positivo al reloj, pero nunca se le debe restar un valor positivo.

Por ejemplo, consideremos tres procesos que se ejecutan en diferentes máquinas, cada una con su propio reloj y velocidad [Figura: 5]:

- El proceso "0" envía el mensaje "a" al proceso "1" cuando el reloj de "0" marca "6".
- El proceso "1" recibe el mensaje "a" cuando su reloj marca "16".
- Si el mensaje acarrea el tiempo de inicio "6", el proceso "1" considerará que tardó 10 marcas de reloj en viajar.
- El mensaje "b" de "1" a "2" tarda 16 marcas de reloj.
- El mensaje "c" de "2" a "1" sale en "60" y llega en "56", tardaría un tiempo negativo, lo cual es imposible.
- El mensaje "d" de "1" a "0" sale en "64" y llega en "54".
- Lamport utiliza la relación "ocurre antes de":
  - Si "c" sale en "60" debe llegar en "61" o en un tiempo posterior.
  - Cada mensaje acarrea el tiempo de envío, de acuerdo con el reloj del emisor.
  - Cuando un mensaje llega y el reloj del receptor muestra un valor anterior al tiempo en que se envió el mensaje:
    - El receptor adelanta su reloj para que tenga una unidad más que el tiempo de envío.

Figura 5: Ejemplo de tres procesos cuyos relojes corren a diferentes velocidades. El algoritmo de Lamport corrige los relojes.



El algoritmo cumple nuestras necesidades para el tiempo global, si se hace el siguiente agregado:

- Entre dos eventos cualesquiera, el reloj debe marcar al menos una vez. Si un proceso envía o recibe dos mensajes en serie muy rápidamente, avanza su reloj en (al menos) una marca entre ellos.
- Dos eventos no deben ocurrir exactamente al mismo tiempo. Esto se puede solucionar asociando el número del proceso en que ocurre el evento y el extremo inferior del tiempo, separados por un punto decimal. Asi, si ocurren eventos en los procesos 1 y 2, ambos en el tiempo 40, entonces el primero se convierte en 40.1 y el segundo en 40.2.

Con este algoritmo se puede asignar un tiempo a todos los eventos en un sistema distribuido, con las siguientes condiciones:

- Si “a” ocurre antes de “b” en el mismo proceso, “C(a) < C(b)”.
- Si “a” y “b” son el envío y recepción de un mensaje, “C(a) < C(b)”.
- Para todos los eventos “a” y “b”, “C(a)” es distinto de “C(b)”.

El algoritmo de Lamport nos da una manera de obtener un orden total de todos los eventos en el sistema.

3.2. Relojes Físicos

El algoritmo de Lamport proporciona un orden de eventos sin ambigüedades, pero:

- Los valores de tiempo asignados a los eventos no tienen porqué ser cercanos a los tiempos reales en los que ocurren.
- En ciertos sistemas (Ej.: sistemas de tiempo real), es importante la hora real del reloj:

- Se precisan relojes físicos externos (más de uno por razones de eficiencia y redundancia).
- Se deben sincronizar:
  - Con los relojes del mundo real.
  - Entre sí.

La medición del tiempo real con alta precisión no es sencilla. Desde la invención de los relojes mecánicos, el tiempo se ha medido astronómicamente. Se considera el día solar al intervalo entre dos tránsitos consecutivos del sol, donde el tránsito del sol es el evento en que el sol alcanza su punto aparentemente más alto en el cielo. El segundo solar se define como 1 / 86.400 de un día solar. Como el período de rotación de la tierra no es constante, se considera el segundo solar promedio de un gran número de días. Con la invención del reloj atómico, fue posible medir el tiempo de manera mucho más exacta y en forma independiente de todo el ir y venir de la Tierra. Los físicos retomaron de los astrónomos la tarea de medir el tiempo y definieron al segundo como el tiempo que tarda el átomo de cesio 133 para hacer 9.192.631.770 transiciones. Se tomó este número para que el segundo atómico coincida con el segundo solar promedio de 1958.

La **Oficina Internacional de la Hora en París (BIH)** recibe las indicaciones de cerca de 50 relojes atómicos en el mundo y calcula el tiempo atómico internacional (**TAI**).
Como consecuencia de que el día solar promedio (**DSP**) es cada vez mayor, un día TAI es 3 mseg menor que un DSP constituyendo un problema. La BIH resolvió el problema introduciendo segundos de salto para hacer las correcciones necesarias para que permanezcan en fase. Esta corrección da lugar a un sistema de tiempo basado en los segundos constantes TAI, pero que permanece en fase con el movimiento aparente del Sol. Se lo llama Tiempo Coordinado Universal(**UTC**).

El **Instituto Nacional del Tiempo Estándar** (**NIST**) de EE. UU. y de otros países:

- Operan estaciones de radio de onda corta o satélites de comunicaciones.
- Transmiten pulsos UTC con cierta regularidad establecida (cada segundo, cada 0,5 mseg, etc.).
- Se deben conocer con precisión la posición relativa del emisor y del receptor:
  - Se debe compensar el retraso de propagación de la señal.
  - Si la señal se recibe por módem también se debe compensar por la ruta de la señal y la velocidad del módem.
  - Se dificulta la obtención del tiempo con una precisión extremadamente alta.

3.2.1. Algoritmos de sincronización de relojes físicos

Si se dispone o no de un receptor UTC, la sincronización del tiempo se realiza de la siguiente forma:

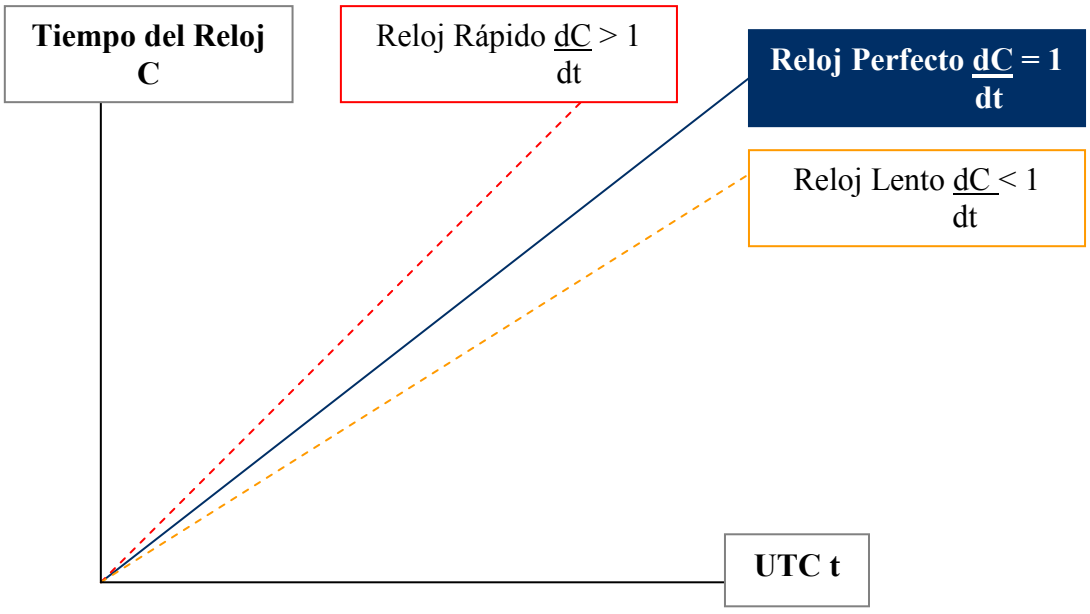
- Si una máquina tiene un receptor de UTC:
  - Todas las máquinas deben sincronizarse con ella.
- Si ninguna máquina tiene un receptor de UTC:
  - Cada máquina lleva el registro de su propio tiempo.

- Se debe mantener el tiempo de todas las máquinas tan cercano como sea posible.

Se propusieron muchos algoritmos para lograr la sincronización, aunque todos los algoritmos tienen el mismo modelo subyacente del sistema, que se describirá a continuación. Se supone que cada máquina tiene un cronómetro que provoca una interrupción "h" veces por segundo. Cuando el cronómetro se detiene, el manejador de interrupciones añade "1" a un reloj de software. El reloj de software mantiene un registro del número de marcas (interrupciones) a partir de cierta fecha acordada con anterioridad; al valor de este reloj se lo llama "C".
Cuando el tiempo UTC es "t", el valor del reloj en la máquina "p" es "Cp(t)":

- Lo ideal sería que "Cp(t)" = "t" para toda "p" y todo "t", ósea que su derivada "dC/dt" debería ser "1".
- Lo real es que los cronómetros no realizan interrupciones exactamente "h" veces por segundo:
  - Poseen un error relativo de aproximadamente 10^{-5} .
  - El fabricante especifica una constante "r" llamada tasa máxima de alejamiento que acota el error.
  - El cronómetro funciona dentro de su especificación si existe una constante "r" y se cumple[Figura: 6]:
    - $1 - r <= dC / dt <= 1 + r$.

Figura: 6. No todos los relojes marcan a la velocidad correcta.



Si dos relojes se alejan de UTC en la dirección opuesta, en un instante dt después de que fueron sincronizados podrían estar tan alejados como 2r dt. Para garantizar que no difieran más de d, se deben volver a sincronizar (en software) al menos cada d / 2r segundos. Los diferentes algoritmos se diferencian en la forma precisa en que se realiza esta resincronización.

A continuación se presentaran algunos algoritmos.

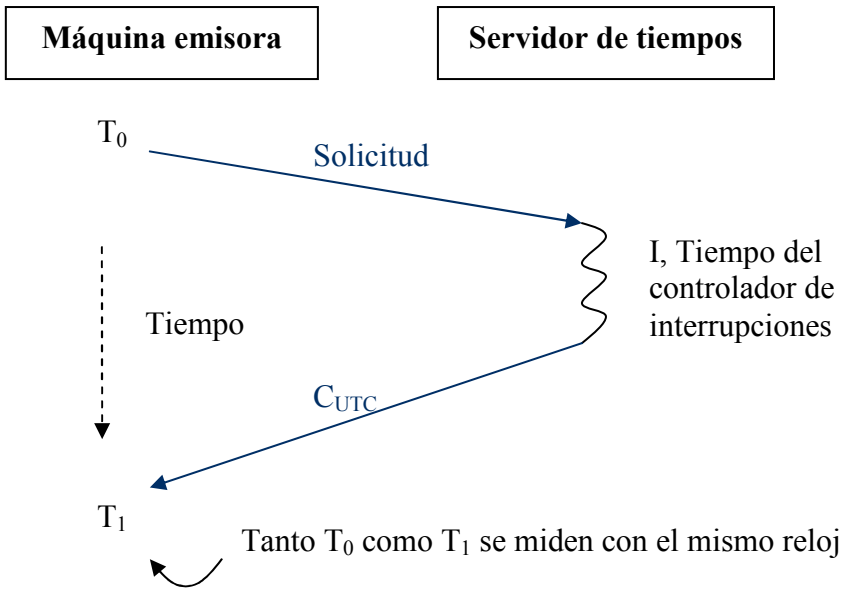
3.2.1.1. Algoritmo de Cristian

Es adecuado para sistemas en los que:

- Una máquina tiene un receptor UTC, por lo que se la llama servidor del tiempo.
- El objetivo es sincronizar todas las máquinas con ella.

Cada máquina envía un mensaje al servidor para solicitar el tiempo actual, periódicamente, en un tiempo no mayor que d / 2r segundos. El servidor del tiempo responde prontamente con un mensaje que contiene el tiempo actual "C_{UTC}" [Figura: 7].

Figura 7: Obtención de la hora actual por medio de un servidor de tiempo.



Como primera aproximación, cuando el emisor obtiene la respuesta puede hacer que su tiempo sea "C_{UTC}". Un problema es que el tiempo nunca debe correr hacia atrás. "C_{UTC}" no puede ser menor que el tiempo actual "C" del emisor. También notemos que el tiempo que tarda el servidor en responder al emisor es distinto de cero. Para resolver este problema el algoritmo de Cristian intenta medir este tiempo de respuesta, teniendo en cuenta que:

- La atención del requerimiento en el servidor de tiempos requiere un tiempo del manejador de interrupciones.
- También se debe considerar el tiempo de transmisión.

El cambio del reloj se debe introducir de manera gradual, pj:

- Si el cronómetro genera 100 interrupciones por segundo:
  - Lo normal seria que cada interrupción añada 10 ms al tiempo.
  - Si quiero atrasar solo agregaría 9 ms.

- Si quiero adelantar agregaría 11 ms.

La corrección por el tiempo del servidor y el tiempo de transmisión se hace midiendo en el emisor:

- El tiempo inicial (envío) "T_0".
- El tiempo final (recepción) "T_1".
- Ambos tiempos se miden con el mismo reloj.

El tiempo de propagación del mensaje será $(T_1 - T_0) / 2$. Si el tiempo del servidor para manejar la interrupción y procesar el mensaje es "I":

- El tiempo de propagación será $(T_1 - T_0 - I) / 2$.

Para mejorar la precisión:

- Se toman varias mediciones.
- Se descartan los valores extremos.
- Se promedia el resto.

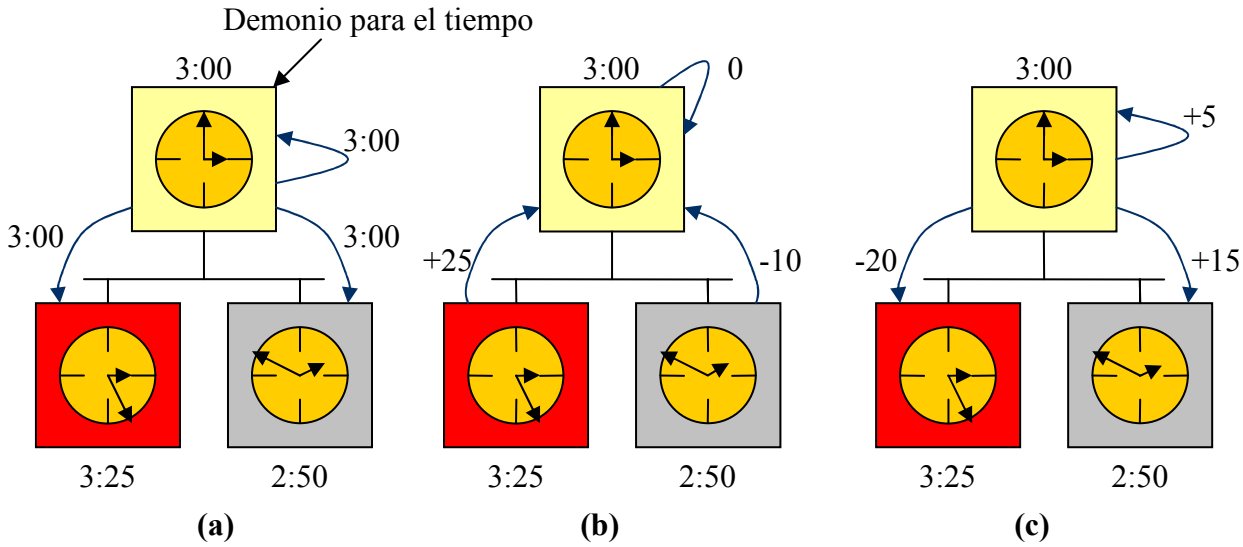
El tiempo de propagación se suma al tiempo del servidor para sincronizar al emisor cuando éste recibe la respuesta.

3.2.1.2. Algoritmo de Berkeley

En el algoritmo de Berkeley el servidor de tiempo:

- Es activo (un demonio para el tiempo), a diferencia del algoritmo de Cristian en el cual el servidor de tiempo es pasivo.
- Realiza un muestreo periódico de todas las máquinas para preguntarles el tiempo.
- Es adecuado cuando no se dispone de un receptor UTC.
- Con las respuestas:
  - Calcula un tiempo promedio, seleccionando solo a aquellos relojes que no difieran entre si una cierta cantidad especifica.
  - Calcula la desviación de cada cliente.
  - Envía la corrección a aplicar en cada cliente. [Figura: 8].

Figura 8: (a) El demonio para el tiempo pregunta a todas las demás maquinas la diferencia de tiempo con respecto a su reloj. (b) Las maquinas contestan. (c) El demonio para el tiempo les indica la forma de ajustar sus relojes.



3.2.1.3. Algoritmo de Promediación

Los anteriores son algoritmos centralizados. Ahora presentaremos un algoritmo descentralizado. Una clase de algoritmos descentralizados divide el tiempo en intervalos de resincronización de longitud fija:

- El i -ésimo intervalo:
  - Inicia en "T_0 + i R" y va hasta "T_0 + (i + 1) R".
  - "T_0" es un momento ya acordado en el pasado.
  - "R" es un parámetro del sistema.

Al inicio de cada intervalo cada máquina transmite el tiempo actual según su reloj. Debido a la diferente velocidad de los relojes las transmisiones no serán simultáneas. Luego de que una máquina transmite su hora, inicializa un cronómetro local para reunir las demás transmisiones que lleguen en cierto intervalo "S". Cuando recibe todas las transmisiones se ejecuta un algoritmo para calcular una nueva hora para los relojes. El algoritmo más sencillo consiste en promediar los valores de todas las demás máquinas. Otra variante es descartar los valores extremos antes de promediar (los "m" mayores y los "m" menores). Esto se realiza como mecanismo de autodefensa contra "m" relojes fallidos que envían mensajes sin sentido. Una mejora al algoritmo considera la corrección de cada mensaje al añadir una estimación del tiempo de propagación desde la fuente. Esta estimación se puede hacer a partir de la topología conocida de la red o al medir el tiempo que tardan en regresar ciertos mensajes de prueba.

3.3. Conclusión

La sincronización del tiempo es fundamental para aquellos modelos de sincronización de procesos que tienen al tiempo como elemento principal de sus algoritmos, en

especial en los sistemas distribuidos. En los sistemas centralizados también lo es, pero no es tan complejo conseguir que todos los procesos tengan una misma referencia: el reloj compartido. En cambio en los sistemas distribuidos cada computadora tiene su propio reloj, por lo cual sincronizarlos no es una tarea sencilla.
Lamport señalo que la sincronización de relojes no necesita ser absoluta. Por una parte, si dos procesos no interactúan, no tienen ninguna necesidad de que sus relojes estén sincronizados, porque la falta de sincronización no sera observable y no causará problemas. Por otra parte, lo que normalmente importa no es que todos los procesos estén sincronizados según una misma hora exacta, sino que todos estén de acuerdo sobre el orden en el que se suceden los eventos que se producen. Lamport invento un mecanismo llamado reloj lógico, el cual es simplemente un contador software monótono creciente, cuyo valor no necesita tener ninguna relación concreta con ningún reloj físico. En cada computadora hay un reloj lógico que se utiliza para poner marcas de tiempo a los eventos que se producen.
El algoritmo de Cristian esta pensado para entornos en los que se dispone de una computadora que está sincronizada con una hora UTC (servidor), y el resto de las computadoras se sincronizan con ella (clientes). Si bien el esquema general del algoritmo presenta problemas (posibilidad de fallo del servidor, problemas de malfuncionamiento o fraude por parte del servidor, escalabilidad), algunos son solucionados con extensiones del mismo. El algoritmo de Berkeley está pensado para entornos en los que no se dispone de ningún receptor UTC, y lo único que se pretende es que todas las computadoras se mantengan sincronizadas con una misma hora. Los algoritmos de sincronización de relojes físicos que hemos visto son centralizados, porque se basan en servicios ofrecidos por un único servidor. Un algoritmo totalmente distribuido es el algoritmo de Premediación. Como hemos visto, resulta difícil sincronizar múltiples relojes distribuidos para que todos ellos mantengan una única hora estándar con la precisión deseada.

Para algunos entornos en los que no se requiere una sincronización exacta con una hora externa de referencia (tiempo UTC), en lugar de tratar con relojes físicos, se trabaja con relojes lógicos. En estos relojes solamente tiene importancia el orden de los eventos, no la medida del momento exacto en el que se producen.
En este capitulo vimos los principales esquemas de algoritmos que resuelven el problema de la sincronización de relojes, de los cuales se derivan otros algoritmos. Los algoritmos presentados tienen ventajas y desventajas, en general el uso de los mismos depende del ambiente en el cual serán utilizados.

4. ESTADO DEL ARTE: Sincronización del Tiempo en Linux

4.1. Network Time Protocol (NTP) en Linux

Anteriormente vimos los diferentes modelos en los cuáles se basan muchos de los algoritmos que realizan la sincronización de relojes en un ambiente distribuido. Ahora vamos a explicar la solución que viene incorporada en Linux. Network Time Protocol (NTP) [31] es un protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes en redes con latencia variable. En realidad NTP son tres cosas: un programa de software NTP (un demonio en Linux); un protocolo que intercambia valores de tiempo entre servidores y clientes; y un conjunto de algoritmos que procesan los valores de tiempo para avanzar o retrazar el reloj del sistema. Éste protocolo basado en un sistema cliente-servidor. Provee a los clientes con tres productos fundamentales: clock offset, round-trip delay y referencia de dispersión. El offset especifica la diferencia entre la hora del sistema local y el reloj externo de referencia. Round-trip delay especifica las latencias de tiempo medidas durante la transferencias de paquetes dentro de la red. La referencia de dispersión de tiempo especifica el máximo número de errores asociados con la información de tiempo recibido de un reloj externo. Las estampas de tiempo utilizadas por NTP consisten en un segundo de 32-bit y una parte fraccional de 32-bit, dando con esto una escala de 2^{32} segundos (136 años), con una resolución teórica de 2^{-32} segundos (0.233 nanosegundos).

NTP es uno de los protocolos de Internet más viejos que siguen en uso (desde antes de 1985). Fue diseñado originalmente por David L. Mills de la Universidad de Delaware, el cual lo sigue manteniendo, en conjunto con un equipo de voluntarios. NTP utiliza un sistema de jerarquía de estratos de reloj, en donde los sistemas de estrato 1 están sincronizados con un reloj externo tal como un reloj atómico. Los sistemas de estrato 2 de NTP derivan su tiempo de uno ó más de los sistemas de estrato 1, y así sucesivamente.

NTP utiliza el Algoritmo de Marzullo[32] (usado para seleccionar fuentes para una estimación precisa de tiempo de un numero de fuentes de tiempo ruidosas) con la escala de tiempo UTC[33]. Incluyendo soporte para características como segundos intercalares o segundo adicional[34](son necesarios para mantener los estándares sincronizados con los calendarios civiles). Se disponen de varias versiones del protocolo cuya precisión esta limitada al reloj de hardware de la PC y al mecanismo de ajuste. La ultima versión de NTP (NTPv4) puede mantenerse sincronizado con una diferencia máxima de 10 milisegundos (1/100 segundos) a través de Internet, y puede llegar a acercarse hasta 200 microsegundos o más en redes de área local con condiciones ideales. Hay una forma menos compleja de NTP que no requiere almacenar la información respecto a las comunicaciones previas que se conoce como Protocolo Simple de Tiempo de Red ó **SNTP**. Ha ganado popularidad en dispositivos incrustados y en aplicaciones en las que no se necesita una gran precisión.

Desde el punto de vista de Linux, parte del paquete de funciones NTP ya viene implementado en el propio kernel. Debido a que se movió el código de ajuste del reloj y la predicción de fase/frecuencia del espacio de usuario al espacio del kernel, con el

motivo de mejorar la precisión del reloj.

Hasta aquí ya tenemos una implementación de la sincronización de relojes en un ambiente distribuido, ahora vamos a centrarnos en la sincronización de relojes en un ambiente multicore.

4.2. Sincronización en ambientes multicore

En los modelos anteriores de sincronización de relojes que hemos visto en el capitulo 3, se consideraba la sincronización en un ambiente distribuido. En el inicio del capitulo 4 se describió la implementación utilizada por Linux. Ahora analizaremos la sincronización en las arquitecturas x86 con procesadores multicore. En las arquitecturas x86 con procesadores multicore disponemos de varias fuentes posibles para ser utilizadas como orígenes de reloj, el entorno de estas arquitecturas posee en general las siguientes características:

- Existe solo una computadora y no varias.
- Existe un reloj común, que es el de la propia computadora, por lo que el tiempo no es ambiguo (notar que Linux solo permite la selección de relojes validos, explicaremos esto posteriormente).
- Al usar el mismo reloj los procesos coinciden en el orden en que ocurren los eventos.
- Los relojes disponibles poseen diferentes precisiones.

Estos relojes disponibles en las arquitecturas x86 con procesadores multicore no necesitan sincronizarse internamente en la computadora porque hay un único reloj disponible por cada posible origen de reloj. Una vez que un reloj es seleccionado por Linux es utilizado sin inconvenientes por los procesos que lo utilizan. Sin embargo, existe un origen de reloj llamado el TSC el cual no es único en las arquitecturas x86 con procesadores multicore, sino que existe físicamente un TSC por cada core del procesador (en general). Esto causa problemas porque Linux utiliza al TSC como si fuera único cuando lo selecciona como origen de reloj, pero esto no es cierto en los procesadores multicore. Por lo tanto, se haya la necesidad de sincronizar los TSC.

Los modelos anteriores de sincronización de relojes en sistemas distribuidos no pueden ser aplicados directamente para sincronizar los TSC, porque tenemos diferencias en el entorno de aplicación. A continuación veremos la historia del TSC en las arquitecturas x86.

4.3. Inconvenientes del TSC para obtener una hora sincronizada

Con la mejora de la tecnología aplicada en el desarrollo de microprocesadores (duplicación de componentes dentro de un mismo microprocesador) surgieron problemas con el uso del TSC como origen del reloj para la interpolación del tiempo. A continuación se explicarán los problemas principales del uso del TSC, más exactamente los problemas encontrados cuando se ejecuta una instrucción de lectura sobre el TSC (rdtsc).

4.3.1. Problemas del uso del rdtsc en PCs semi-modernas

Con el advenimiento de las CPUs multi-core/hyperthreaded[6](sistemas con múltiples núcleos/sistemas que pueden ejecutar múltiples threads simultáneamente), y sistemas operativos que "hibernan", el RDTSC puede no ofrecer resultados confiables. Tomemos como ambiente de explicación un ambiente multicore, en el cual tenemos varios núcleos dentro de un mismo paquete físico de microprocesador y además en cada núcleo tenemos un registro TSC.

Tengamos en cuenta que el modulo de manejo del tiempo de Linux requiere que las lecturas realizadas sobre el dispositivo que se use como cronometro, deben retornar valores crecientes sino se producirá valores de lecturas de hora erróneos. Debido a que el modulo de manejo del tiempo utiliza estos valores para calcular el tiempo actual. Mas adelante se explicará el funcionamiento del modulo del manejo del tiempo. Otra consideración a tener en cuenta es que al haber un TSC por núcleo, la instrucción rdtsc leerá del TSC del núcleo sobre el cual se este ejecutando.

Problema 1: valores discontinuos:

En un sistema monoprocesador cuando un proceso ejecuta una lectura sobre el TSC, este le retornara un valor X, en la siguiente lectura le retornara un valor Y, y siempre el valor X va a ser menor o igual a Y (en general menor). Esto se repite independientemente de si el proceso que lee el TSC es el mismo u otro diferente, por lo que los valores de lectura son siempre incrementales.

En cambio, en un sistema multicore, no se garantiza la sincronización de sus contadores de ciclos (TSCs) entre cores. Dado que los valores de sus respectivos TSCs no están coordinados. Por ejemplo, supongamos que tenemos un procesador con dos núcleos y contamos con dos threads ejecutándose uno en cada núcleo; cada uno de estos threads lee el TSC del core donde se están ejecutando. Por lo que tenemos el siguiente caso de lecturas consecutivas:

Instante de lectura	Thread 1: lee valor	Instante de lectura	Thread 2: lee valor
1	5	2	16
3	**10**	4	20
5	**15**	6	25

Notemos que las lecturas sobre el TSC del mismo core son incrementales pero las lecturas sobre diferentes TSC no lo son, dándonos una lectura de 5, 16, 10, 20, 15, 25; lo cual producirá un fallo en el calculo de la hora actual.

Problema 2: Variabilidad de la frecuencia de la CPU:

Antes se asumía que la frecuencia de la CPU era fija durante toda la vida del programa. Sin embargo, con las modernas tecnologías de administración de energía, esta asumpción es incorrecta. Ahora la CPU puede variar su frecuencia.

Supongamos el caso monoprocesador con variación de frecuencia, el manejador del

TSC en el inicio del sistema calcula la frecuencia del TSC (cantidad de ciclos por segundo). Posteriormente una vez calculada la frecuencia del TSC podemos saber a cuantos nanosegundos equivalen cada ciclo del TSC. Ahora si cambiamos de frecuencia en tiempo de ejecución, el mecanismo de manejo del tiempo no ejecuta funciones que actualicen las variables del manejador del TSC. De modo que se pueda actualizar la nueva equivalencia entre ciclos del TSC y su equivalente en nanosegundos. Esto produce lo siguiente: Supongamos un proceso P1 que quiere saber cuanto tiempo paso desde que comienza a correr hasta que finaliza; para esto lee el TSC 2 veces; una en el inicio y otra en el fin del proceso. Finalmente llama a una función que calcula la cantidad de nanosegundos a la que equivale un ciclo de TSC, por lo que multiplica este valor por la cantidad de ciclos que recibe como parámetro obteniendo el tiempo que tarda el proceso en ejecutarse.

P1:

```
c1 = leer TSC;
... //cosas que hace el proceso
c2 = leer TSC;
cantidadDeCiclosTotales = c2-c1;
tiempo = calcularEquivalenciaEnNanosegundos(cantidadDeCiclosTotales);
```

En caso de que la frecuencia del TSC sea constante la variable “tiempo” obtendrá un valor valido (dado que en el inicio del sistema se calculo la frecuencia y con ello la equivalencia entre ciclos y nanosegundos). Pero en caso de que se halla cambiado la frecuencia del TSC durante la ejecución del proceso P1, el valor calculado en el inicio del sistema para la equivalencia entre ciclos y nanosegundos, ya no es valida, produciéndose un valor de tiempo erróneo. Si se aumenta la frecuencia un ciclo de TSC representara una menor cantidad de nanosegundos, si se disminuye la frecuencia un ciclo de TSC representara una mayor cantidad de nanosegundos. Por lo que el sistema operativo debe estar atento a estos cambios de frecuencia.

En el caso de un ambiente multicore, al tener varios TSCs, los cuáles pueden estar corriendo a diferentes frecuencias, tenemos que agregar que además estos pueden cambiar de frecuencia dinámicamente en tiempo de ejecución, agravando el problema.

Problema 3: Mecanismo de gestión de energía:

En realidad esto seria el porque del problema 2, actualmente los microprocesadores proveen de un mecanismo para la gestión de energía, el cual permite poner a diferentes niveles de frecuencia a un mismo procesador incluso detenerlo. A esto se les dice niveles de profundidad para dormir a la CPU, necesario para reducir el consumo de energía y el calor disipado de la CPU.
El problema principal del ahorro de energía en cuando al uso del TSC por parte del modulo del manejo del tiempo es que el TSC ahora se puede detener totalmente, lo cual podría producir lecturas de TSC con valores no crecientes incluso sobre el mismo núcleo. Mas específicamente el tiempo transcurrido durante el cual el TSC estaba apagado se pierde, debido a que al resumir el TSC este comienza a contar de nuevo.

En conjunto estos tres problemas producen que ya no se pueda obtener valores fiables de la hora a menos que se cierre el programa para utilizar una sola CPU. Incluso

entonces, la velocidad de la CPU puede cambiar debido al poder de las medidas de ahorro adoptadas por el sistema operativo o el BIOS, o el sistema puede hibernar y más tarde reanudar (reajustando el TSC).

4.3.2. Solución parcial al problema del TSC en PCs semi-modernas:

Los pre-últimos procesadores de Intel leen; para el TSC; la tasa máxima de los procesadores independientemente de la tasa de la CPU en funcionamiento. Si bien esto hace que el mantenimiento del tiempo sea más consistente, puede producir ciertos puntos de referencia, donde una cierta cantidad de tiempo es gastada para que el sistema operativo cambie al procesador de una tasa de reloj baja a una tasa más alta. Esto tiene el efecto de hacer que las cosas requieran más ciclos de procesador de lo que normalmente necesitarían. Esta solución complico la gestión por parte del sistema operativo pero aún no solucionaba la posible detención total del TSC.

4.3.3. Solución parcial al problema del TSC en PCs modernas: TSC invariante/constante

Los últimos procesadores soportan una nueva característica, referida como TSC invariante. Esta característica indica que el TSC corre a una tasa constante en todos los P-, C- y T-estados del procesador en cuanto a administración de energía[35], independientemente de la frecuencia actual del procesador. De esta forma el sistema operativo puede utilizar al TSC para los servicios de temporización. Con esta solución física la lectura del TSC es mucho más eficiente y no incurre en la sobrecarga asociada a tener que estar cambiando de la frecuencia actual a la frecuencia máxima cuando se lee el TSC; lo que sucedía en la solución anterior. La configuración de la tasa constante del TSC esta identificada por la bandera constant_tsc en Linux cuando se consulta por la información de la CPU (/proc/cpuinfo).

4.3.4. Implementación en varios procesadores

Las familias de procesadores Intel incrementan el contador del TSC en forma diferente [36]:

- Para los procesadores Pentium M (familia [06H], los modelos [09H, 0DH]); para procesadores Pentium 4, procesadores Intel Xeon (familia [0FH], los modelos [00H, 01H, o 02H]), y para la familia de procesadores P6. El TSC se incrementa con cada ciclo del reloj interno del procesador. El ciclo del reloj interno del procesador es determinado por la razón/cociente del reloj del bus y del reloj del core actual. Las transiciones de la tecnología para la gestión de energía pueden incidir también en el reloj del procesador.

- Para algunos procesadores Pentium 4, procesadores Intel Xeon (familia [0FH], los modelos [03H y superior]); para procesadores Intel Core Solo y Core Duo (familia [06H], modelo [0EH]). Para el procesador Intel ® Xeon ™ serie 5100 y procesadores Intel Core Duo 2 (familia [06H], modelo[0FH]); para los procesadores Intel Xeon (display_model [17H]). Para procesadores Intel Atom (familia [06H], display_model [1CH]). El TSC se incrementa en una tasa constante. Aquella tasa puede ser seteada por la máxima razón del reloj del core y del reloj del bus del procesador o puede

ser seteada por la máxima frecuencia resuelta en la que el procesador es arrancado. La máxima frecuencia resuelta puede diferir de la máxima frecuencia calificada para el procesador, dado que la medición en fábrica se hace para un ambiente determinado.

Por ejemplo, en procesadores Intel core i7, una aplicación puede hacer uso del TSC corriendo a una frecuencia constante cuando se este en modo de salvado de energía. Por ejemplo: razón de operación base* 133.33Mhz (reloj base), durante los C-estados [37][38]. La razón de operación base puede ser leído desde los bits del registro MSR_PLATFORM_INFO[15:8]; esta es la razón de la frecuencia a la que corre el TSC invariante (ósea el valor del multiplicador). El valor del reloj base también puede ser leído desde otro registro especifico del modelo; dependiendo de la microarquitectura del microprocesador estos valores pueden encontrarse en diferentes ubicaciones.

TSC invariante= razón de operación base* velocidad escalable del reloj del bus del sistema

La familia de procesadores AMD también incrementan el contador TSC de forma diferente:

- Los procesadores AMD hasta la familia K8 (en otros términos, hasta los procesadores sempron) siempre incrementan el contador de sello de tiempo por cada ciclo de reloj. Por lo tanto, la capacidad del cambio de frecuencia por parte del modulo de gestión de energía, puede cambiar el número de incrementos del TSC por segundo. Entonces los valores pueden salir de sincronización entre los diferentes cores de los procesadores o en el mismo sistema [39].
- Desde la familia 10h (Barcelona/Phenom) los chips AMD disponen de una característica llamada constant TSC, que puede ser manejada por la velocidad del Hyper-Transport (bus de alta velocidad) o el más alto estado P. Estos estados P son estados operacionales del core, indican cuando se esta ejecutando algún trabajo y el core esta con determinado estado de voltaje y frecuencia.

Por ejemplo, en el caso del TSC constante, existe una sola fuente de reloj en el puente norte [40] para todos los TSCs en un mismo procesador y estos contadores se incrementan al mismo paso. Esto permite que el contador de ciclos proporcione valores monotonicamente incrementados a una tasa/ritmo constante independientemente de la frecuencia del core, y del estado de performance [41].

4.4. Arquitectura Linux para el Manejo del tiempo

Linux debe llevar varias actividades relacionadas con el tiempo. Por ejemplo, el kernel periódicamente:

- Actualiza el tiempo transcurrido desde el inicio del sistema.
- Actualiza la hora y la fecha.
- Determina, para cada CPU, cuánto tiempo el proceso actual se ha estado ejecutando, y se apropia de este si se ha sobrepasado el tiempo asignado al

mismo.
- Actualiza las estadísticas de utilización de recursos.
- Verifica si el intervalo de tiempo asociado con cada temporizador de software ha transcurrido.

La arquitectura de manejo del tiempo de Linux es el conjunto de estructuras de datos del kernel y funciones relacionadas con el flujo del tiempo. Actualmente, las máquinas con multiprocesador basados en Intel tienen una arquitectura de manejo del tiempo que es ligeramente diferente de la arquitectura de manejo del tiempo de máquinas monoprocesador:

- En un sistema monoprocesador, todas las actividades del mantenimiento del tiempo son activadas por interrupciones emitidas por un temporizador global (el PIT o el HPET).
- En un sistema multiprocesador, todas las actividades generales (como el manejo de temporizadores de software) son activadas por las interrupciones levantadas por un temporizador global. Mientras que la actividades específicas de CPU (como el monitoreo del tiempo de ejecución del proceso actualmente en ejecución) se activan por las interrupciones levantadas por el temporizador del LAPIC. Aunque también pueden ser emuladas como broadcast entre los procesadores a partir del temporizador global.

La distinción entre los dos casos es un poco difusa. Por ejemplo, algunos Sistemas Multi-Procesador (SMP) tempranos basados en procesadores Intel 80486 no tienen LAPICs. Por otra parte, existen sistemas monoprocesador que tienen un LAPIC y un I/O APIC, por lo que el kernel puede utilizar la arquitectura de manejo del tiempo SMP. Sin embargo, para simplificar la descripción, no se van a discutir estos casos híbridos y se mantendrá a las dos arquitecturas de manejo del tiempo "puras".

4.4.1. El kernel mide el paso del tiempo en diferentes maneras

A continuación se describirán las diferentes maneras en las cuáles el kernel mide el tiempo, aunque pueden existir variaciones de estas medidas [42]:

- Wall time (o tiempo real): Es el tiempo y fecha actual del mundo real, que es el tiempo que uno lee sobre el reloj de la pared. Los procesos usan el wall time cuando interfacean con el usuario o se sella con tiempo un evento.
- Process time (tiempo de proceso): Es el tiempo que ha consumido un proceso (todos sus threads), de forma directa en código de espacio de usuario, o indirecta vía el trabajo del kernel en medio del proceso. Sirve para profiling y estadísticas. Por ejemplo, medir cuanto tomo una operación dada. Wall time no es adecuada para medir procesos por la naturaleza multitarea de Linux, el tiempo del proceso puede ser mucho menor que el wall time para una operación dada. Un proceso también puede gastar significantes ciclos de espera para una E/S (particularmente en entradas del teclado).
- Monotonic time (tiempo monotónico): Esta fuente de tiempo es estrictamente incrementada linealmente. Linux utiliza el tiempo desde el inicio del sistema. El wall time puede cambiar-por ejemplo, porque el usuario lo seteo, y porque el sistema continuamente ajusta el tiempo por desvíos- y adicional imprecisión puede ser introducida, por decir, saltos de

segundos. El tiempo desde el inicio del sistema, por otra parte, es una representación del tiempo deterministica e incambiable. El aspecto importante de una fuente monotonica de tiempo no es su valor actual, sino la garantía de que la fuente del tiempo sea estrictamente incrementada linealmente. De manera que sea totalmente usable para calcular la diferencia en tiempo entre dos muestreos. En Linux existe una representación de ajuste ntp del tiempo en orden a medir como corregir el tiempo manteniéndolo monotónico.
- Monotonic raw/puro: Es similar a monotonic time solo que no se considera el ajuste ntp (no se realizan ajustes de frecuencia).

Estas medidas de tiempo pueden ser representadas en uno de dos formatos:

- Tiempo relativo: Este es un valor relativo a algún punto, tal como el instante actual: por ejemplo, 5 segundos desde ahora, o hace 10 minuto. Monotonic time/Monotonic raw, son ideales para calcular tiempo relativo.
- Tiempo absoluto: Esto representa el tiempo sin algún punto de referencia: por ejemplo, 25 de marzo de 1968. En Linux el tiempo absoluto se representa como el numero de segundos desde la época, que es definida como 00:00:00 UTC (Universal Time, Coordinated) del 1 de enero de 1970. Wall time es ideal para medir tiempo absoluto.

4.4.1.1. Estructura de datos para el tiempo

A continuación, se describirán las diferentes estructuras de datos utilizadas para representar el tiempo:

- Para representar una precisión de segundos se utiliza la estructura time_t, en la cual se define una variable que representa segundos como long.
- Para representar una precisión de microsegundos se utiliza la estructura timeval, en la cual se definen dos variables, tv_sec (que define segundos como long) y tv_usec (que define microsegundos como long).
- Para representar una precisión de nanosegundos se utiliza la estructura timespec, en la cual se definen dos variables, tv_sec (que define segundos como long) y tv_nsec (que define nanosegundos como long).
- Un tipo para el tiempo de los procesos: El tipo clock_t representa las marcas del reloj, es un long. Dependiendo de la interface, las marcas de aquel clock_t significa la frecuencia del temporizador actual del sistema (HZ) o CLOCKS_PER_SEC.

4.4.1.2. Obteniendo el actual tiempo del día (Get Time of Day)

Desde menor a mayor precisión las funciones para obtener el tiempo del sistema serían [43]:

- time(): Obtiene el tiempo con una resolución de segundos.
- gettimeofday():Obtiene el tiempo con una resolución de microsegundos.
- clock_gettime(): POSIX provee esta interface para obtener el tiempo de una fuente de tiempo específica enviada como parámetro. Se definen varias constantes que representan a fuentes de reloj con un determinado

comportamiento y precisión, pero solo requiere CLOCK_REALTIME_COARSE que es el reloj de tiempo real de todo el sistema. Los demas valores son: CLOCK_MONOTONIC_COARSE, CLOCK_PROCESS_CPUTIME_ID, CLOCK_THREAD_CPUTIME_ID, CLOCK_MONOTONIC_RAW, CLOCK_MONOTONIC, CLOCK_REALTIME Y CLOCK_BOOTTIME. Por lo tanto, dependiendo del reloj utilizado se permite una resolución de nanosegundos (por ejemplo, utilizando CLOCK_PROCESS_CPUTIME_ID).

Si bien todas estas llamadas son definidas como llamadas al sistema existen llamadas virtuales al sistema. Por ejemplo el gettimeofday virtual que evita los procesos y verificaciones relacionadas a las llamadas al sistema, eliminando el cambio de contexto entre contexto de usuario y de sistema.

4.4.2. Variables principales utilizadas en la arquitectura del manejo del tiempo

Una marca/tick es una unidad arbitraria para medir un intervalo de tiempo del sistema; es utilizada para coordinar todas las operaciones de la CPU y la memoria. A cuantos milisegundos representa una marca depende del sistema operativo, en Linux tenemos sistemas con diferentes valores como por ejemplo 4ms por marca.
Consideremos que el TSC cuenta los ciclos de CPU no las marcas del sistema operativo y en caso de que el microprocesador tenga la característica que hace que el TSC funcione a una tasa constante independientemente de los cambios de frecuencia entonces ya no contara los ciclos de CPU.

A continuación enumeraremos las variables principales que participan en el esquema de manejo del tiempo de Linux:

1-La variable **jiffies** es un contador que almacena el número de marcas transcurridas desde que el sistema se inició. Esta es incrementada en uno cuando una interrupción temporizada ocurre, es decir, en cada marca; su extensión es la variable **jiffies_64**.

2-La variable **xtime** almacena la hora y fecha actuales, es una estructura de tipo timespec y tiene dos campos:

- tv_sec: Almacena el número de segundos que han transcurrido desde la medianoche del 1 de enero de 1970 (UTC).
- tv_nsec: Almacena el número de nanosegundos que han transcurrido en el último segundo (su valor oscila entre 0 y 999.999.999).

La variable xtime se actualiza normalmente una vez en una marca. Los programas de usuario obtienen la hora y fecha actuales de la variable xtime.

3-La variable **raw_time** es similar a xtime salvo que esta variable no es ajustada por el mecanismo NTP.

4-El seqlock **xtime_lock** evita las condiciones de excepción que podrían ocurrir debido a accesos concurrentes a la variable xtime. Recordemos que xtime_lock también protege

a la variable jiffies_64, en general, este seqlock se usa para definir varias regiones críticas en la arquitectura de manejo del tiempo.

5-La variable **wall_to_monotonic** es del mismo tipo timespec como xtime, y esencialmente almacena el número de segundos y nanosegundos que se añadirán a xtime con el fin de obtener un flujo monotónico (creciente) de tiempo. De hecho, tanto los saltos de segundos y sincronización con relojes externos pueden cambiar de repente los campos de xtime de modo que ya no son monotonicamente incrementados (a veces el núcleo necesita una verdadera fuente monotónica de tiempo).

4.4.3. Características Generales del nuevo sistema del tiempo de Linux

4.4.3.1. Generic Time Of Day

Por un momento ampliemos el análisis de nuestra arquitectura de trabajo, arquitecturas x86, y pensemos en otras arquitecturas tales como SPARC, ALPHA, etc. Linux da soporte para el manejo del origen del reloj (clocksource) en varias arquitecturas; aunque dicho soporte se dificulta debido a que una modificación en una arquitectura puede tener que ser replicada para cada una de las demás arquitecturas. O peor aun, existe una gran cantidad de duplicación de código para las diferentes arquitecturas con respecto al manejo del origen del reloj [44].
El subsistema genérico del tiempo del día (GTOD) reescribe el código del manejo del origen del reloj. Lo que se hizo fue mover el código que se repite en las diferentes arquitecturas a un subsistema genérico de gestión (constituido por un conjunto de funciones, variables y constantes). Además se dejo las porciones de código dependientes de la arquitectura simplemente reducidas a los detalles de hardware de bajo nivel del origen del reloj.

Con lo cual tenemos lo siguiente:

- Las arquitecturas dispondrán de manejadores de los orígenes de reloj (drivers para sus relojes) para sus respectivas arquitecturas.
- Existirá un subsistema de gestión de dichos orígenes de reloj, el cual concentrará todo el código común a las diferentes arquitecturas.
- Los orígenes de reloj podrán registrarse en el subsistema para su posible uso.
- Los manejadores de los orígenes de reloj deberán implementar una interface provista por el subsistema genérico.
- El subsistema seleccionará al origen de reloj para la interpolación del tiempo basándose en la calidad del mismo (cada reloj sabe que calidad tiene, y la especifica como un valor numérico de uno de sus atributos).
- Se podrán agregar y retirar orígenes de reloj en tiempo de ejecución o incluso cambiar de origen de reloj manualmente.
- El código para obtener el tiempo del día (TOD) esta completamente separado del manejo de las marcas periódicas.

4.4.3.1.1. Administración del origen del reloj

El número y la resolución de los disponibles orígenes de reloj varían mucho sobre las diferentes arquitecturas, algunos de ellos son solo detectables en tiempo de ejecución.

GTOD provee un excelente código base para la abstracción del origen del reloj [45], para esto se consideraron los siguientes detalles:

- Selección de resolución.
- Soporte para sistemas sin marcas y con marcas incluyendo una integración limpia dentro del código de manejo de interrupción.
- Usabilidad de orígenes de reloj para diferentes propósitos (manejo del tiempo, planificación de eventos de alta/baja resolución).
- Aritmética 32-bits vs. 64-bits.

Una capa de abstracción y su asociada API (Interfaz de Programación de Aplicación, es el conjunto de funciones y procedimientos que pueden ser utilizados por otro software como una capa de abstracción) son requeridas para establecer un subsistema de código común para administrar varias fuentes de reloj. Dentro de este subsistema, cada fuente de reloj mantiene una representación del tiempo según sus valores (ej: tantos ciclos del reloj A equivalen a X unidades de tiempo). Los nanosegundos son la unidad de valor de tiempo de una fuente de reloj. Esta capa de abstracción permite al usuario seleccionar en medio de un rango de dispositivos de hardware disponibles. La infraestructura generada incluye, por ejemplo, funciones matemáticas (dependientes de propiedades específicas de cada dispositivo de hardware) para convertir valores de tiempo específico de cada origen de reloj dentro de unidades de tiempo de nanosegundos. La centralización de esta funcionalidad permite al sistema compartir significativamente más código entre las diferentes arquitecturas.

En el Anexo B se muestra un ejemplo de la estructura del clocksource.

4.4.3.1.2. Sincronización de reloj

Las fuentes de reloj manejadas por cristales tienden a ser imprecisos debido a variaciones en la tolerancia de los componentes y factores ambientales, tales como cambios en la temperatura. Lo que resulta en una tasa de marcas de reloj ligeramente diferente y de esta forma, en cuanto al tiempo, valores de reloj diferentes en diferentes computadoras. También consideremos que el mecanismo de manejo del tiempo esta limitado a una determinada precisión en cuanto a la representación de valores. Al producirse estos efectos sobre el sistema evidentemente el reloj del sistema también será afectado. Terminando en una ganancia o perdida de varios segundos por semana si el reloj corre libremente sin algún tipo de compensación [45].

El protocolo del tiempo de red (NTP) y los últimos mecanismos de sincronización basados en GPS/GSM (Sistemas de Posicionamiento Global/Sistema Global para las Comunicaciones Móviles) permiten la corrección de los valores del tiempo del sistema. Y también permiten corregir los desvíos de frecuencia de la fuente de reloj a partir de una fuente de reloj de referencia. La corrección del valor es aplicado al valor del origen de reloj hardware para el incremento monotónico cuando sea posible en caso contrario se aplica al reloj software del sistema. Esta es una funcionalidad opcional que puede ser aplicada cuando se dispone de una fuente de reloj de referencia. El subsistema NTP es el que provee una manera para que el kernel mantenga la pista del desvío del reloj y el cálculo para ajustarlo. Este subsistema es el que viene implementado en Linux por default.

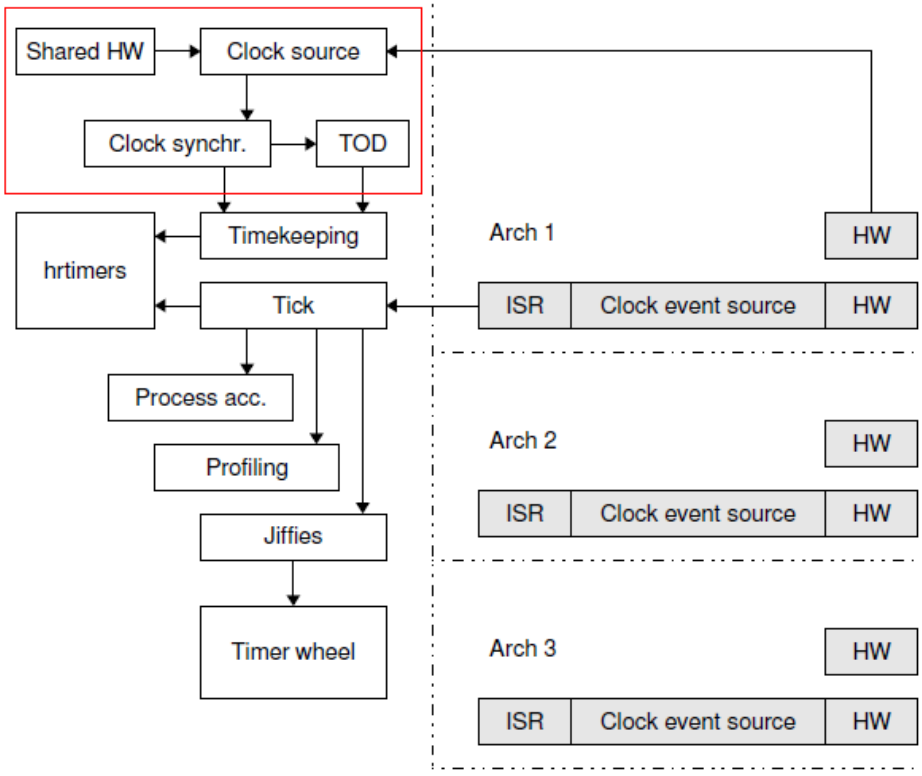
4.4.3.1.3. Representación del Time-of-Day

La interface genérica para la representación del time-of-day debe compensar los desvíos como un offset/desfase del valor del origen del reloj, y representar el time-of-day (calendario o tiempo de reloj de pared) como una función del valor del origen del reloj. Este reloj de software corregido siempre tendrá un crecimiento monotónico.
El offset de desvío y los parámetros de la función convierten al valor del origen del reloj en un valor de reloj de pared que puede ser seteado por una interacción manual o bajo el control de un software para sincronización con orígenes de tiempo externos (ej: NTP).
La API del time-of-day hace uso de una eventualmente corregida frecuencia de las fuentes de tiempo para implementar la interface "humanamente legible" (traducción de ciclos de reloj a nanosegundos y viceversa).

En el Anexo C se muestra un ejemplo de la estructura principal para el manejo del tiempo en Linux.

El siguiente grafico conceptual muestra GTOD (el administrador de los orígenes de reloj) y su relación con otros componentes que participan en el manejo del tiempo.

Figura 10: Esquema conceptual del GTOD.



Donde Arch x es una arquitectura especifica, con su código de fuente de eventos de reloj (para la rutina de servicio de interrupción) duramente asociada con dispositivos individuales de hardware (hasta que se aplique el siguiente subsistema clockevent). TOD (funciones para la obtención del tiempo del día) e ISR (rutinas de servicio de interrupción) son capas de abstracción genérica. Los servicios relacionados al reloj están vinculados con los siguientes módulos:

- Timekeeping (manejo del tiempo).

- TOD (funciones para la obtención del tiempo del día).
- Process acounting (para conocer los recursos del sistema usados, alocación de recursos, monitorización del sistema y mínimamente trazar los comandos ejecutados por los usuarios).
- Profiling (para recolectar datos estadísticos de usos de los recursos).
- Timer Whell (traza del tiempo usando las marcas de temporización periódicas).
- Hrtimers (forma parte de la API para temporizadores de alta resolución) [46].

4.4.3.2. API clockevents

4.4.3.2.1. Análisis del antiguo sistema del tiempo

La implementación antigua del temporizador relacionado al manejo de eventos poseía una configuración y selección de la fuente de eventos de una forma cableada/fija dentro del código dependiente de la arquitectura. Esto se traduce en código duplicado en todas las arquitecturas y hace que sea muy difícil cambiar la configuración del sistema para utilizar dispositivos de eventos de interrupción que no sean los ya incorporados en la arquitectura. Otra consecuencia del diseño es que es necesario tocar todas las implementaciones específicas de la arquitectura con el fin de proporcionar nuevas funcionalidades como temporizadores de alta resolución o marcas dinámicas.
Analizando las implementaciones del timer_interrupt() en diversas arquitecturas se llega en general a una relación 1:1 de copiar y pegar código similar con algunos controles específicos de la arquitectura. Muchos de estos controles están relacionados a la variedad de las fuentes de eventos de reloj que puede ser desconocida en tiempo de compilación [47].

El análisis del temporizador en relación a la funcionalidad obtiene los siguientes componentes:

- Manejo de marcas.
- Update_process_times (actualización de tiempos de procesos).
- Profiling (para datos estadísticos).
- Extensiones: eventos no basados en marcas (temporizadores de alta resolución, marcas dinámicas).

Estas funcionalidades pueden ser manejadas por una variedad de entornos de hardware:

- Única fuente de eventos manejando todo.
- Fuentes de eventos múltiples.
- Origen de eventos único para las marcas y uno o más orígenes de eventos por CPU para otras funcionalidades.

Esto introdujo una gran cantidad de controles #ifdef y macros mágicos dispersos en el código. Si queremos por ejemplo agregar nuevas funcionalidades, como temporizadores de alta resolución, se incrementa el número de controles de manera significativa en todas las posibles combinaciones de manejo.

4.4.3.2.2. Modificación del sistema del tiempo (incorporación de una capa de abstracción)

El subsistema de eventos de reloj (o clockevents) proporciona una solución genérica para administrar los dispositivos de eventos de reloj y su uso para varias funcionalidades del kernel manejadas por eventos de reloj. El objetivo del subsistema de eventos de reloj es reducir al mínimo el código dependiente de la arquitectura relacionado a eventos de reloj y permitir una fácil adición y utilización de nuevos dispositivos de eventos de reloj. La idea básica consiste en mover la asignación de la fuente de eventos desde el tiempo de compilación a nivel de arquitectura hacia una decisión en tiempo de ejecución. Esta solución proporciona una funcionalidad genérica bajo el manejador del servicio de interrupción, que es ampliamente dependiente del hardware. Desde el manejador del evento de reloj propio de cada arquitectura se llama a un manejador genérico, el cual realiza las funciones genéricas del manejador; validas para todas las arquitecturas.

4.4.3.2.3. Detalles del subsistema clockevents

Clockevents crea una API para manejar dispositivos quienes pueden entregar interrupciones en un tiempo específico en el futuro. Esto posibilita que dichos dispositivos de eventos de reloj sean usados para planificar el próximo evento de interrupción/interrupciones. El próximo evento en la actualidad esta definido como periódico, con su periodo definido en tiempo de compilación. La API sigue las capacidades de cada temporizador (resolución y si se puede tener interrupciones one-shot (de una vez) o periódicas) y ofrece una simple interface para armar el temporizador. Esta API se define en el núcleo central, con solo un manejador de bajo nivel que queda en el código de cada arquitectura especifica [48][49]. También se puede decir que la API es pequeña y dispone de diferentes servicios de notificación de dispositivo de evento de reloj y soporte para suspender o resumir.

La capa de administración de reloj almacena un puntero a función en la estructura del descriptor del dispositivo, quien tiene que ser llamado desde el manejador a nivel de hardware. Mas exactamente, el manejador de la fuente de eventos debe implementar un descriptor (provisto por la API clockevents). Este descriptor además de especificar las características propias de la fuente de eventos también tendrá un puntero al manejador genérico, el cual contendrá las actividades comunes del manejo de una interrupción, de todas las arquitecturas diferentes.
Los dispositivos de eventos de reloj son registrados por el código de arranque del kernel dependiente de la arquitectura o en tiempo de inserción del modulo. Cada dispositivo de eventos de reloj llena una estructura de datos con parámetros de propiedades específicas del reloj y llamadas a funciones.
El gestor de eventos de reloj decide, a través del uso de específicos parámetros de propiedad del reloj, el conjunto de funciones de sistema de un dispositivo de eventos de reloj que serán usados para soporte. Esto incluye la distinción de dispositivos de evento global por sistema y por CPU.

El resultado final es que el kernel tiene ahora los medios para consultar y utilizar las funciones del temporizador en una manera independiente de la arquitectura. Con el mecanismo de clockevents, se hace posible soportar verdaderos temporizadores de alta resolución. Cuando tal temporizador es solicitado, todo lo que se necesita es elegir un

dispositivo de clockevent adecuado y armar éste para el tiempo deseado. Estos dispositivos pueden entregar interrupciones con un alto grado de precisión, con lo que resulta que los temporizadores del núcleo, también, pueden ofrecer alta precisión- una característica que es de clara utilidad para los usuarios de tiempo real (entre otros).

La capa de administración asigna una o más de las siguientes funciones a un dispositivo de evento de reloj:

- Sistema global de marcas periódicas (actualización de jiffies).
- Update_process_times (actualización de tiempos de proceso) de la CPU local.
- Profiling (para obtener datos estadísticos) de la CPU local.
- Próximo evento de interrupción de la CPU local (modo no periódico).

Se pueden dividir a los dispositivos de eventos de reloj en dos tipos:

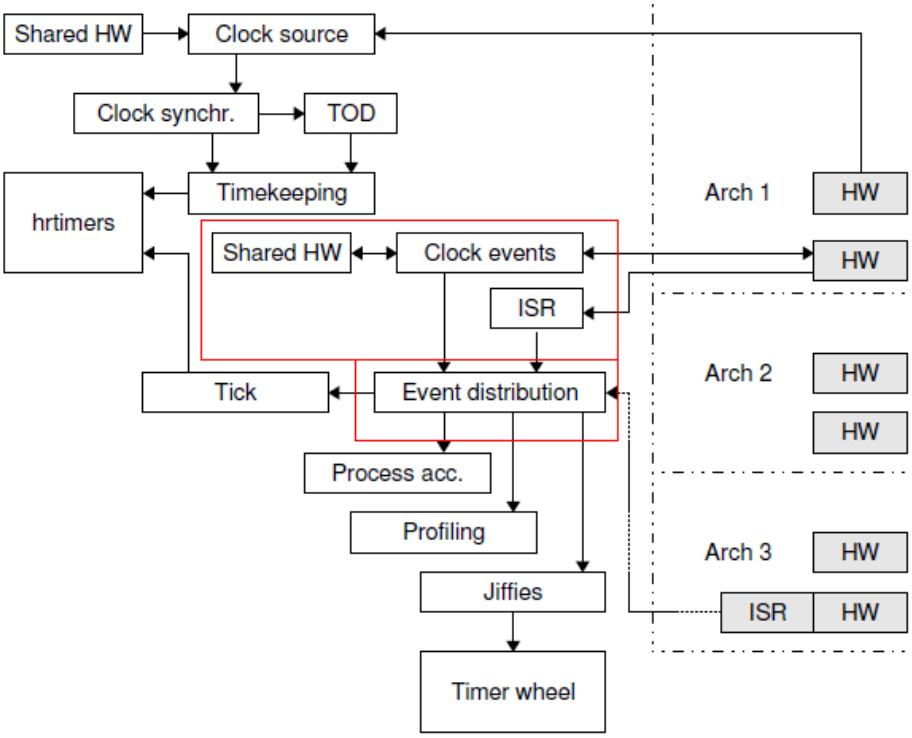
- Los dispositivos de evento global a nivel de sistema, son usados para las marcas periódicas de Linux.
- Los dispositivos de evento por CPU son usados para proveer funcionalidades para la CPU local tales como monitorización de usos de recursos, profiling, y temporizadores de alta resolución.

En el Anexo D se muestra un ejemplo de la estructura del descriptor del dispositivo de evento de reloj para el mantenimiento del tiempo en Linux.

También, existe una estructura tick_device para cada CPU y una para el dispositivo broadcast. La del dispositivo broadcast sirve por ejemplo (cuando es configurado) en caso de aquellos microprocesadores en los cuáles sus LAPICs pueden apagarse cuando entran en el estado C3 de administración de energía. En modo broadcast el evento de temporización se podría reenviar a cada CPU (para que hagan sus tareas respectivas asociadas a la temporización del LAPIC); aunque este modo según respectivos análisis indican que produce una sobrecarga y baja de rendimiento. También para que una fuente de eventos pueda ser considerada como posible fuente broadcast, ésta no debe tener como una de sus características CLOCK_EVT_FEAT_C3STOP. Esto último indicaría que la fuente de eventos es afectada por este estado de administración de energía (la posible detención de la fuente). Por ejemplo, el HPET y el PIT no poseen esta característica por lo que pueden ser fuentes de eventos de tipo broadcast.

El siguiente grafico conceptual muestra GTOD (gestión de los orígenes del reloj), Clockevents (gestión de las fuentes de eventos) y sus relaciones con otros módulos.

Figura 11: Esquema conceptual del GTOD y Clockevents.



4.4.4. Algunas consideraciones

A continuación se indicaran algunas consideraciones a tener en cuenta cuando se realiza el análisis teórico y del código fuente del modulo del manejo del tiempo de Linux:

Cuando se haga referencia al reloj del sistema, esto implicará la referencia al reloj de software del sistema; en este caso la variable xtime.

Los cálculos están en complemento a dos (un numero negativo se produce por invertir todos sus bits y sumarle uno al resultado final).

Como no seria conveniente modificar la frecuencia del oscilador del reloj hardware (para ajustar el tiempo en Linux) seria mejor o equivalente modificar un parámetro de la representación en la fuente del reloj, en este caso del mult. Se sabe que la frecuencia por segundo medida en nanosegundos se puede aproximar por los parámetros de mult y shift de una fuente de reloj; al modificar mult estaríamos o acelerando o disminuyendo la frecuencia del reloj, en cierta forma ajustándolo.

La escala del tiempo NTP es mantenida utilizando 64 bits enteros donde los 32 bits superiores representan los segundos enteros y los restantes 32 bits inferiores representan las fracciones de segundo teniendo una resolución de alrededor de 200 picosegundos. Por ello en el código fuente de Linux se pasa de escala a escala utilizando desplazamientos de 32 bits.

Podríamos decir que en el código fuente de Linux se trabaja en cuatro escalas:

- Escala normal (la que conocemos ns, us, ms, ...).
- Escala de reloj (producido por el campo shift en su representación).
- Escala ntp (producido por su shift=32).
- Escala de error (es igual a la mitad de la escala del reloj).

Esto se realiza porque en el sistema se trabajan con variables enteras y no flotantes entonces una buena forma es escalar la representación (multiplicar). Por ello en el código fuente a veces se menciona nanosegundos en escala ntp o nanosegundos en escala tsc; cuando en realidad no son nanosegundos como se conocen sino son una medida mucho mas pequeña. Pero como se trabaja en escalas se les dice nanosegundos; aunque quizás sea conveniente decirles valor en escala ntp, valor en escala tsc, etc.

4.5. Conclusión

En este capitulo vimos como se implementa la sincronización del tiempo en Linux a través del NTP, cuando se dispone de un entorno distribuido. Luego analizamos la sincronización de relojes en ambientes multicore, encontrándonos con un origen de reloj llamado TSC que requería sincronización debido a la forma que tiene Linux de manejar a los posibles orígenes de reloj. El esquema de manejo del tiempo de Linux maneja al reloj seleccionado como si fuera único, pero el TSC no es único en ambientes multicore. Luego, se describieron los problemas y las posibles soluciones parciales al problema de la sincronización del TSC a través de los años. Posteriormente se describió la actual arquitectura de Linux para el manejo del tiempo, desarrollándose dos subsistemas que lo manejan (GTOD y clockevent). Estos dos subsistemas son abstracciones genéricas conformadas por un conjunto de funciones, variables y constantes.
Tengamos en cuenta que el problema de la sincronización del TSC no fue resuelto completamente (en términos generales de una solución aplicada a cualquier arquitectura x86), solo existieron y existen soluciones parciales (para subconjuntos de arquitecturas x86). También, notemos que el TSC es importante en las arquitecturas x86 debido a que es el reloj con la mejor resolución que se puede obtener. Por eso encontrar una solución general lo mas optima posible es un fin.
Finalmente, podemos decir que la transformación del esquema de manejo del tiempo de Linux fue un trabajo de limpieza de código y generalización de funciones notable.

5. ANÁLISIS DE LAS FUNCIONES PRINCIPALES DEL SUBSISTEMA DEL MANEJO DEL TIEMPO

En este capitulo explicaremos las funciones principales que intervienen en las principales actividades del modulo del manejo del tiempo, su conocimiento es necesario debido a que parte de este esquema se utilizará en la nueva solución propuesta por este trabajo. Dividiremos el análisis en los siguientes subtemas:

- Registro del notificador de marcas.
- Inicialización del subsistema de manejo del tiempo.
- Manejador de una interrupción temporizada.
- Llamada a la función gettimeofday.

En cada subtema de análisis se presentará un resumen de las llamadas principales. Luego, se explicará cada una de ellas. En el Anexo E se puede encontrar el pseudocódigo para facilitar la lectura respectiva al flujo de las funciones como así también se eliminarán detalles específicos para concentrar la atención en las actividades principales. Notemos que como es pseudocódigo el texto se describirá como si fuera que existe la función en el kernel pero en realidad lo que existen son funciones equivalentes.

5.1. Registro del notificador de marcas

5.1.1. Resumen

Se registra el notificador en el subsistema clockevents para administrar los eventos relacionados a la marca periódica, por ejemplo cuando se agregan nuevos dispositivos de eventos de reloj.

```
start_kernel()
-----> tick_init();
       -----> clockevents_register_notifier();
              -----> raw_notifier_chain_register();
```

5.1.2. Detalles

En la función principal del kernel se encuentra la función tick_init(), la cual registra el notificador de marcas (ver Anexo A) en el subsistema clockevents para administrar los eventos relacionados a la marca periódica. En si, registra una función sobre el evento de agregación de cada posible origen de eventos de reloj (cuando cada reloj se registra). Dicha función configura el manejador que realiza el código independiente de la arquitectura relacionado al manejador de una interrupción temporizada. La implementación actual del manejador de una interrupción temporizada esta dividida en dos partes; una parte con el código dependiente de la arquitectura y la otra con el código independiente o genérico de la arquitectura. En caso de seleccionar el PIT/HPET este dispositivo se configura en el inicio como periódico; en general la función que lleva a cabo todas las demás funciones independientes de la arquitectura es tick_periodic().

Esta función es llamada por un manejador de mayor nivel tick_handle_periodic() / tick_handle_periodic_broadcast(). Este es el manejador donde empieza el código independiente de la arquitectura y es configurado cada vez que se verifica una nueva registración de un dispositivo fuente de eventos de reloj.

Variables principales utilizadas en este flujo de llamados:

struct notifier_block tick_notifier: Variable que define el notificador de marcas. Notemos que no se inicializa el campo priority; se lo asume 0 (por ser de tipo entero y su valor default es cero); la lista va a estar ordenada de mayor a menor prioridad, y si tienen la misma prioridad el que primero se agrego a la lista se ejecuta antes del segundo.
RAW_NOTIFIER_HEAD(clockevents_chain): Definición de la cadena para la notificación de eventos de reloj.

Funciones principales de este flujo de llamados:

tick_notify(): Notificación sobre dispositivos de eventos de reloj. Cada vez que se agregue un nuevo dispositivo de eventos de reloj el notificador ejecutará la función tick_check_new_device ().
tick_init(): Función que inicializa el control de marcas y registra el notificador con el subsistema clockevents. Registra el bloque que contiene la función interesada en determinados cambios de estados relacionados al dispositivo de eventos.
clockevents_register_notifier: Función que registra un escuchador de cambios de reloj de eventos, agrega a la cadena el bloque notificador que recibe como parámetro, mediante la función **raw_notifier_chain_register().**

5.2. Inicialización del subsistema de manejo del tiempo

A continuación se mostrará como se inicializa el subsistema de manejo del tiempo desde la función principal del kernel:

5.2.1. Resumen

Inicialización del subsistema de manejo del tiempo.

```
start_kernel ()
----->timekeeping_init ();
      ----->clocksource_default_clock();
      ----->timekeeper_setup_internals();
----->time_init();
----->late_time_init()= x86_late_time_init();
      ----->hpet_time_init();
            ----->hpet_enable();
            ----->setup_pit_timer(): Función llamada si no se puede
configurar al HPET.
            ----->setup_default_timer_irq();
      ----->tsc_init();
            ----->init_tsc_clocksource();
                  ----->clocksource_register();

Llamadas dentro de la función hpet_enable():
```

```
----->hpet_clocksource_register();
      ----->clocksource_register();
----->hpet_legacy_clockevent_register():   Función   llamada   si   se
verifica que el HPET dispone de la capacidad de enrutado de irq.
      ----->clockevents_register_device();
            ----->clockevents_do_notify();
```

5.2.2. Detalles

Variables principales utilizadas en este flujo de llamados:

struct timekeeper timekeeper: Variable que mantiene los valores internos para el manejo del tiempo del sistema, se puede ver la estructura en más detalle en el Anexo C.
struct timespec xtime: Tiempo del sistema con estructura que posee dos campos: segundos y nanosegundos.
struct timespec total_sleep_time: Tiempo transcurrido cuando se estuvo en estado suspendido, sirve para que no se produzcan saltos del tiempo monotónico cuando se resume desde el estado suspendido, por lo que el valor de esta variable será agregada a wall_to_monotonic.
struct timespec raw_time: El tiempo monotónico puro (sin ajuste) para el reloj POSIX CLOCK_MONOTONIC_RAW. Es el reloj que no realiza ajuste de tiempo.
struct clock_event_device *global_clock_event: Variable que hará referencia a la fuente de eventos actual.
unsigned int cpu_khz: Indica la frecuencia CPU en khz.
unsigned int tsc_khz: Indica la frecuencia TSC en khz.
int tsc_clocksource_reliable: Indica si el TSC nunca se detiene.
struct clock_event_device pit_ce: Descriptor del dispositivo de evento de reloj PIT.
struct irqaction irq0: Variable cuya estructura mantiene la referencia a la función que maneja la interrupción producida por las marcas periódicas, llamada timer_interrupt.
struct clocksource clocksource_hpet: Definición de la estructura del origen de reloj HPET, un ejemplo de la estructura del clocksource se encuentra en el Anexo B:
struct clocksource *curr_clocksource: Variable que guarda el actual origen de reloj.
struct clock_event_device hpet_clockevent: Descriptor del dispositivo de evento de reloj HPET.

Funciones principales de este flujo de llamados:

start_kernel(): Esta es la función principal del kernel que llama a timekeeping_init(), time_init() y late_time_init().
timekeeping_init(): Inicializa las variables relacionadas al esquema de manejo del tiempo.
time_init(): Configura el campo late_time_init a x86_late_time_init para que a través de esta configuración se retrase el inicio de los temporizadores periódicos hasta después de la llamada a x86_late_time_init(), para permitir que trabaje el ioremap.
late_time_init(): Inicializa el TSC y retarda el inicio de los temporizadores.
timekeeping_init(): Función que inicializa el subsistema clocksource y valores comunes del timekeeping (configura a la variable xtime entre otras cosas).
clocksource_default_clock(): Retorna la estructura clocksource_jiffies que es de tipo clocksource solo que la inicializa con valores para el jiffies; que es el reloj mínimo común que debería funcionar en todos los sistemas. Tiene la misma resolución que la frecuencia de interrupciones del sistema.

timekeeper_setup_internals(): Configura el reloj a utilizar. Es la función que configura los campos del timekeeper para usar un origen de reloj. También, se configuran los campos que representan el tiempo de ajuste NTP del kernel en orden a medir como el tiempo debe ser corregido:
x86_late_time_init(): Inicializa al temporizador por default e inicializa al TSC.
hpet_time_init(): Prueba configurar al HPET dentro del subsistema clocksource. Registra al PIT como temporizador en el subsistema clockevents si no esta disponible el HPET. Registra la irqaction del temporizador.
hpet_enable(): Prueba configurar el temporizador HPET.
setup_pit_timer(): Función que inicializa el factor de conversión, es el factor de multiplicación para el cálculo escalado, quien es usado para convertir valores basados en nanosegundos a marcas de reloj. Y registra la fuente de eventos de reloj PIT en el subsistema clockevents.
setup_default_timer_irq(): Función que asigna la estructura de acción para la interrupción cero, ósea la interrupción producida por las marcas.

hpet_clocksource_register(): Función que registra el origen de reloj HPET. Configura la estructura clocksource_hpet y llama a la función genérica de registro de reloj clocksource_register().
clocksource_register(): Función utilizada para instalar un nuevo origen de reloj. En esta función se encola el reloj (ordenado por reiting). Luego se selecciona al mejor reloj disponible (con respecto al reiting o según su especificación como parámetro de booteo). Finalmente se verifica si se puede detener, en ese caso se lo ubica en otra lista.
hpet_legacy_clockevent_register(): Habilita las interrupciones del HPET, configura el HPET, registra el HPET como fuente de evento y lo asigna como la fuente de eventos global de sistema.
clockevents_register_device: Función utilizada para instalar un nuevo dispositivo de fuente de eventos de reloj.
clockevents_do_notify(): Función que notifica sobre un cambio de reloj de eventos.

tsc_init(): Función que inicializa el TSC.
init_tsc_clocksource(): Inicializa y registra al TSC en el subsistema clocksource.
clocksource_register(): Función que registra orígenes de reloj en el subsistema clocksource.

5.3. Manejador de una interrupción temporizada

La función llamada cada vez que se produce una interrupción de marca es timer_interrupt().

5.3.1 Resumen

Manejador de una interrupción temporizada:

```
timer_interrupt();
----->tick_handle_periodic();
      ----->tick_periodic();
            ----->do_timer(): función llamada si la CPU esta
habilitada para realizar el trabajo a realizar por marca, es porque el
LAPIC también utiliza esta función para manejar su interrupción
temporizada.
```

```
----->update_wall_time();
      ----->logarithmic_accumulation();
```

5.3.2. Detalles

Variables principales utilizadas en este flujo de llamados:

u64 jiffies_64: Variable que guarda las marcas desde el inicio del sistema.

Funciones principales de este flujo de llamados:

timer_interrupt(): Manejador default de la interrupción temporizada para el PIT/HPET, la implementación de este manejador depende de la arquitectura. Primeramente ejecuta las acciones particulares para la arquitectura (ej: optimización). Luego, ejecuta el código genérico para todas las arquitecturas, en el caso de las arquitecturas x86 ejecutará la función tick_handle_periodic() quien a su vez ejecutará tick_periodic().
tick_handle_periodic(): Manejador de eventos para marcas periódicas-modo no broadcast. En el caso de que el dispositivo de eventos de reloj no sea broadcast se ejecutará esta función. Modo broadcast significa lo siguiente: las interrupciones locales pueden ser emuladas por interrupciones broadcast, en caso de que el hardware no soporte temporizadores LAPIC. Notemos que Linux siempre utiliza el último dispositivo de eventos de reloj registrado por lo cual si bien hay una estructura de dispositivo de marcas por CPU solo utiliza un único dispositivo. En caso por ejemplo de que en una parte del código se vuelva a registrar el PIT, en general se configurará de una manera similar a la que fue configurada la primera vez. Salvo que ahora va a estar en posiblemente otra estructura de otra CPU. Aunque esta CPU que primera vez lo registro (u otra en caso de que esa CPU se duerma) es la que va a hacer el trabajo de ejecutar el trabajo de actualización del tiempo del sistema llamando a la función do_timer(). También, se configura el próximo periodo para dispositivos que no tienen modo periódico de interrupción.
tick_periodic(): Realiza la mayor parte de las operaciones a efectuar por marca del sistema. Esta función, es llamada tanto por las interrupciones del LAPIC/modo broadcast como por la interrupción producida por las marcas del sistema. Por lo cual, si la CPU es la que fue designada a manejar las funciones específicas de cada marca solo ella va a ejecutar algo más que las demás CPUs (las actividades generales para todas las CPUs). Por lo que cada CPU debe realizar la actualización del tiempo de proceso y calcular estadísticas pero solo una CPU debe actualizar el reloj del sistema.
do_timer(): Incrementa el contador de marcas del sistema jiffies con ticks (en este caso 1). Actualiza el tiempo del sistema, llamando a update_wall_time(). Calcula la carga del sistema.
update_wall_time(): Función que actualiza el tiempo del sistema. Usa el origen de reloj actual para incrementar el tiempo del sistema, representado por la variable xtime. Calcula la cantidad de ciclos totales a consumir y los va consumiendo, principalmente servido por la función logarithmic_accumulation(). Llamado desde timer_interrupt(), debe mantener un write sobre xtime_lock.

logarithmic_accumulation(): Esta función acumula un intervalo desplazado de ciclos dentro de un intervalo desplazado de nanosegundos. Permitiendo loops de acumulación de O(log). Consume dos a la shift (ciclos de reloj por marca) ciclos de la cantidad de ciclos que faltan procesar. Se puede decir que la granularidad de consumo de marcas es por marca (ej: consumo 1,2,4,8,16 marcas, pero siempre al nivel de marcas como granularidad de proceso). Retorna los ciclos sin consumir. Ejemplo de funcionamiento, cada vez que se produce una marca por ejemplo se producirían x ciclos de reloj. En el caso perfecto se producirá el manejo de la interrupción (cuando se saca la diferencia entre el ultimo ciclo leído y el actualmente leído) cuando pasen x ciclos. Pero que pasa en el caso de producirse el manejo después de más de x ciclos, evidentemente habría que consumir los x ciclos y algún restante, en el caso de sistemas sin marcas esta diferencia puede ser grande. Lo que hace esta función es consumir los ciclos de una manera óptima, en grandes trozos. Si a (ciclos totales a consumir)=x (ciclos por marca) + restante (posiblemente x*n (siendo n un entero)) entonces ésta función consumirá este restante y retornara el restante que no llega a completar la cantidad de ciclos x. Se usan logaritmos en base dos por una cuestión de agilizar cálculos con desplazamientos de bits (multiplicar por 2 a la shift o dividir por 2 a la shift). Por eso a esta función se le envía como parámetro la diferencia entre ambos máximos común multiplicador con base dos de la cantidad de ciclos totales a consumir y de los ciclos de reloj por marca. Debido a que la función en si es compleja, vamos a dividirla en tres partes:
Parte A: En esta parte principalmente vamos a verificar cuantos ciclos hay que consumir, y si la cantidad es menor a la cantidad de ciclos a consumir en un intervalo de marca no se hace nada y se sale.
Parte B: Acumula un intervalo de ciclos a consumir, la cantidad a consumir es restada del total de ciclos a consumir, se actualiza el ultimo ciclo consumido y se actualiza la cantidad de nanosegundos en escala del reloj a consumir (que serán agregados al reloj del sistema).
Parte C: Acumula el tiempo bruto (sin ajuste), esto sirve para el reloj POSIX que es similar al reloj del sistema salvo que no realiza correcciones NTP. Se calculan los nanosegundos a consumir en escala normal y posteriormente se agregan estos nanosegundos al reloj puro sin ajuste.
Parte D: Acumular el error entre el intervalo a consumir de reloj y el del NTP.

5.4. Llamada a la función gettimeofday()

Para obtener la hora actual, las aplicaciones ejecutan la llamada al sistema gettimeofday() quien es implementada por la función sys_gettimeofday(), quien invoca a do_gettimeofday(). Ejemplo de las acciones realizadas por la función do_gettimeofday:

5.4.1 Resumen

Obtención de la hora actual del día:

```
do_gettimeofday();
----->getnstimeofday();
      ----->timekeeping_get_ns();
```

5.4.2. Detalles

Funciones principales de este flujo de llamados:

do_gettimeofday(): Función que retorna el tiempo del día en una estructura timeval, con resolución de microsegundos.
getnstimeofday(): Obtiene el tiempo del día en una estructura timespect, resolución de nanosegundos.
timekeeping_get_ns(): Obtiene los nanosegundos transcurridos desde la última interrupción temporizada

6. SINCRONIZACIÓN ENTRE CORES

En los capítulos anteriores analizamos los respectivos módulos que afectan al esquema del tiempo, ahora ya estamos en condiciones de poder generar un conjunto de soluciones software posibles para el problema de la sincronización en un ambiente multicore. Recordemos que el problema de la sincronización en ambientes multicore, fue debido a la replicación de TSCs en un mismo procesador (uno por cada core), los cuales pueden no estar sincronizados (no corren a la misma velocidad, poseen valores distintos). Como Linux en su esquema de manejo del tiempo asume que cada posible origen de reloj es único, la falta de sincronización entre estos TSCs (cuando al TSC se lo selecciona como origen de reloj del sistema) produce problemas en el modelo de manejo del tiempo actual (reloj no monotónico).
En este capítulo describiremos inicialmente los problemas del TSC en arquitecturas multicore. Luego describiremos un conjunto de soluciones posibles, iniciando con una solución simple y finalizando con una solución final.

6.1. Inconvenientes del TSC en arquitecturas multicore

En los microprocesadores modernos se dispone de un registro llamado time stamp counter (TSC) éste sirvió para obtener datos estadísticos sobre el funcionamiento del microprocesador (monitoreo de su funcionamiento). En cierta forma indicaba la frecuencia del procesador. Con el advenimiento de los procesadores con varios núcleos, se comenzó a duplicar ciertas propiedades físicas de los microprocesadores obteniendo así arquitecturas con un TSC para cada procesador lógico dentro de un mismo paquete físico.

Si bien el TSC desde su concepción no fue diseñado para ser utilizado como posible origen del reloj para la interpolación del tiempo, en Linux se lo utilizaba como tal, dada su resolución de nanosegundos. En los posteriores años nos encontrábamos con diferentes TSCs por cada procesador lógico más diferentes mecanismos de administración de energía y posibilidad de cambio de frecuencia en los procesadores lógicos. Esto produjo que si se utilizaba como origen del reloj al TSC, el subsistema de manejo del tiempo se volvía inestable (crecimiento no monotónico del reloj, inconsistencias en las aplicaciones que hacían uso de los servicios del reloj). Luego el TSC se podría detener y además los valores de diferentes TSCs eran diferentes cuando se cambiaba la frecuencia del procesador lógico. En Linux se lo deshabilito como posible fuente de reloj (aunque se podía utilizar si se aseguraba que el proceso que requería su uso se mantuviese en el mismo procesador lógico).

Desde Intel propusieron una solución física (solucionando según indican los problemas y las ineficiencias de las posibles soluciones a nivel de software), incorporando una nueva característica del procesador llamada constant TSC. La cual indica que el TSC funciona a una tasa constante independientemente de los posibles cambios de frecuencia del procesador. La solución de Intel aparenta ser la más óptima pero se pierde la idea de que el TSC representa la frecuencia del procesador actual, otro efecto secundario es que el mecanismo de administración de energía no podrá optimizar su uso. Dado que siempre correrá y nunca se detendrá. Y principalmente resta por decir que esta solución solo funciona en las arquitecturas que dispongan de dicha característica.

Antes de continuar expliquemos que son los estados idle: A nivel de CPU, se puede controlar la energía utilizada de varias maneras, una de ellas es através de los estados idle o C-estados. Ellos reflejan la capacidad de que un procesador ocioso apague componentes que no necesita para salvar potencia. Cuando un procesador corre en el estado C0, éste esta ejecutando instrucciones. Un procesador que corre en otro C-estado esta ocioso. Cuando más alto sea el número del C-estado, el modo de dormir de la CPU es más profundo: lo que indica que más componentes serán desactivados para salvar potencia. Cuanto más profundamente se duerma la CPU se salva más potencia, pero la desventaja es que este proceso tiene alta latencia (el tiempo que necesita la CPU para volver a despertarse a un estado C0). Algunos estados también tienen submodos con diferentes niveles de salvado de potencia y su correspondiente latencia. El soporte de los C-estados y sus submodos depende del respectivo procesador. Aunque, siempre el estado C1 esta disponible. Algunos de los C-estados son:

- C0: Estado operacional. La CPU esta completamente encendida.
- C1: Halt. Primer estado de ociosidad. Estado donde el procesador no esta ejecutando ninguna instrucción pero típicamente no esta en un estado de potencia bajo. La CPU puede continuar procesando prácticamente sin demora. Todos los procesadores ofrecen este estado.
- C2: Stop-Clock. Estado donde el reloj es detenido vía hardware por este procesador pero mantiene el completo estado de sus registros y caches, para que de esta forma después de reiniciar el reloj pueda inmediatamente reiniciar el procesamiento. Éste es un estado opcional.
- C3: Sleep. Estado donde el procesador detiene todos los relojes internos de la CPU y no necesita mantener su cache coherente. Despertar de este estado toma considerablemente más tiempo que volver del estado C2. Éste estado es opcional. Actualmente Intel desaconseja su uso.

6.2. Modelo de sincronización en microprocesadores sin característica constant tsc

En lo que resta de este capitulo pretenderemos proveer de varias soluciones para poder utilizar al TSC como posible origen del reloj en microprocesadores sin la característica constant tsc con varios núcleos; la complejidad de las mismas iran creciendo a medida que el reloj sea más preciso.

6.2.1. Descripción general

- Se va a generar un nuevo reloj de software (a través de un driver), manteniéndose una estructura global de representación del origen de reloj. Este nuevo reloj mantendrá la hora actual del sistema (similar a raw_xtime; ósea sin ajuste, por lo que va a ser de crecimiento lineal) llamada tiempo_sistema aunque su inicio partirá de la variable xtime.
- Una CPU se va a encargar del inicio del reloj (calibrándolo e iniciando las variables necesarias) y posteriormente se actualizará el reloj en cada marca.
- Se proveerá de una interface virtual a través del sistema de archivos virtuales ./proc para que cualquier aplicación pueda obtener el tiempo del nuevo reloj con solo leer del archivo virtual. Con lo que no se requerirá realizar una

llamada al sistema para obtener el tiempo del día (evitando demoras en verificaciones).

6.2.2. Modelo simple de diseño básico del reloj de software

- Definir una variable para el reloj global del sistema.
- Definir una variable para mantener el tiempo del sistema.
- Definir una variable para mantener el último ciclo leído.
- Generar una tarea para inicializar el TSC.
    - calibrar el TSC.
    - inicializar estructura del TSC y variables necesarias.
    - generar una tarea para inicializar el tiempo.
        - inicializar el tiempo del sistema, copiando desde la variable más precisa (xtime).
        - inicializar el punto de último ciclo leído.
        - generar una tarea para actualizar el tiempo que será ejecutada con un tiempo de espera mínimo de una marca.
- Generar la interface de comunicación en ./proc llamada “tiempo”.

Luego cada vez que se lea del archivo virtual ./proc/tiempo se ejecutara la función:

- getns_tiempo_del_dia: retorna el tiempo del nuevo reloj, la función leer_tsc() controlará que siempre los ciclos leídos del reloj sean mayores o iguales al último ciclo leído.

6.3. Un reloj más fiel

Analizando los módulos de administración de energía[50][51][52][53][54] y cambios de frecuencia[55][56][57] se le puede otorgar una mayor precisión al nuevo reloj dándole soporte para tales características, generando diferentes posibles combinaciones a solucionar. En lo que sigue, se describirá la idea general de la solución y cada una de las soluciones posibles, indicando si es para monoprocesador o multicore y también si maneja cambios de frecuencia o detenciones totales del TSC.

6.3.1. Núcleo general

Las actividades comunes de las distintas soluciones son las siguientes:

- Se genera un reloj de software.
- Se calcula la frecuencia de la CPU de inicio y se configura (al igual que el subsistema clocksource).
- Se inicia el trabajo que provee el servicio de obtención de hora para el nuevo reloj.
- Las actualizaciones del tiempo del sistema se manejan con un trabajo periódico como mínimo después de una marca, este trabajo se ejecuta en el procesador de inicio.

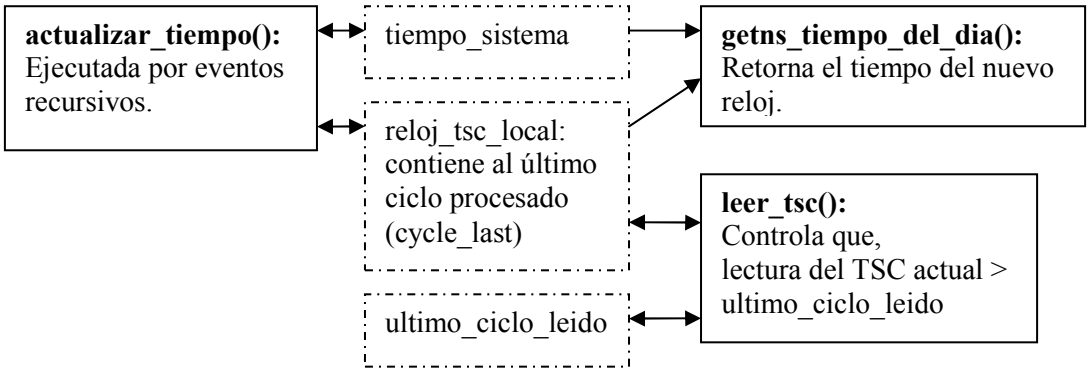
6.3.2. Generalizando la solución

El conjunto de soluciones siguiente muestra como a partir de una solución genérica simple se evoluciona a una solución genérica más compleja pero lo mas precisa posible. En dicha evolución, aparecen aspectos tales como la arquitectura del microprocesador (monoprocesador o multicore, manejo de cambios de frecuencia, posible detención del TSC) y la gestión aplicada por parte de Linux. La manera en la que se van abarcando estos aspectos permite la afinación de la solución inicial propuesta.

6.3.2.1. Monoprocesador-Multicore e independiente de cambios de frecuencia e idles profundos:

Solución simple, corresponde a la solución explicada anteriormente. En esta solución las líneas principales del código, son las que hacen referencia en definitiva a cuando se lee del TSC del core actual. Si el ciclo leído es menor al último ciclo leído (independientemente del TSC de lectura) se retorna el último ciclo leído sino el actualmente leído, quien además debe ser mayor al último ciclo procesado por última vez, en caso contrario se retorna el último ciclo procesado. Esta solución es adecuada para aplicaciones que no posean requerimientos precisos del tiempo, y que tan solo pretendan un crecimiento monotónico del tiempo.

Esquema conceptual de funcionamiento:



6.3.2.2. Monoprocesador con manejo de idles profundos:

En este caso el modulo de administración de energía administra estados de ociosidad y va pasando de estado en estado (a nivel de profundidad de estado idle). En las arquitecturas x86 de 64 bits existen notificadores de inicio-fin de estados idle. Utilizando estos notificadores se debe registrar una función interesada en tales eventos (mas específicamente solo en los eventos en los cuáles el estado idle detiene el TSC). Luego se debe cronometrar con un reloj de referencia cuanto tiempo se estuvo en estado idle y finalmente agregar este tiempo al tiempo del sistema; en detalle:

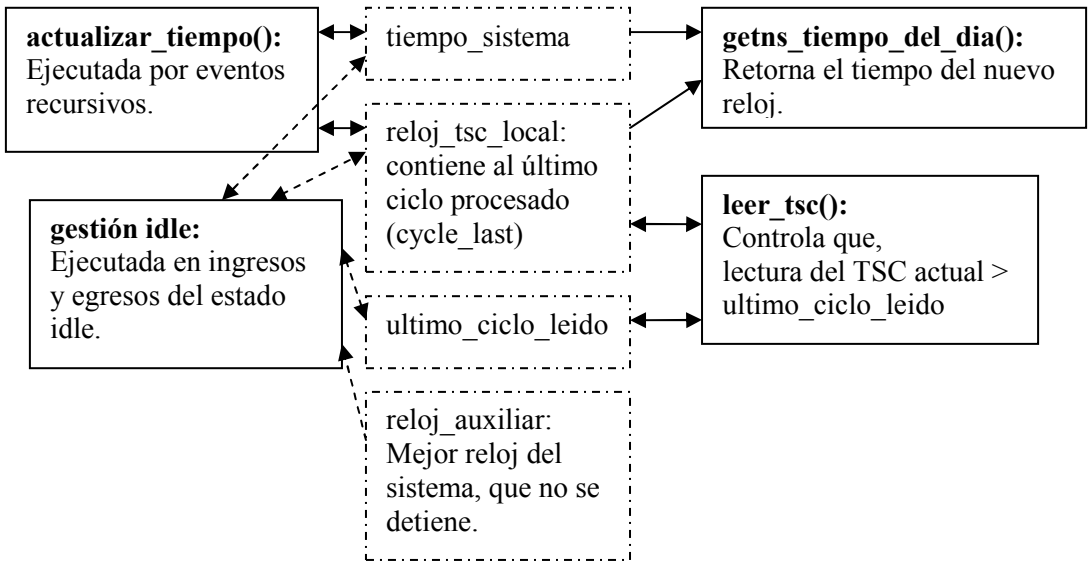
Entrada idle:

- Actualización del tiempo del sistema.
- Tomar como reloj de referencia al mejor reloj del sistema (usando GTOD) con el cual se calculara el tiempo inicial.

Salida idle:

- Preguntar si no existe inicio de idle (para manejar los fin de idle "significativos", dado que cuando se maneja una interrupción éste emite un fin de idle sin haber entrado en estado idle). Si ese es el caso entonces terminar sino tomar el tiempo final según el reloj de referencia, calcular el tiempo total en el cual estuvo en idle, agregar este tiempo calculado al tiempo del sistema y resetear el ultimo ciclo procesado leído.

Esquema conceptual de funcionamiento:



Esta solución es adecuada para aplicaciones que no posean requerimientos precisos del tiempo, y que solo pretendan un crecimiento monotónico. La misma es similar a la anterior pero además mejora el manejo del tiempo en estados idle profundos, gracias a que toma los tiempos en el ingreso y egreso de este estado y posteriormente los agrega al nuevo reloj. De esta forma, utilizando al mejor reloj del sistema que no se detiene, se miden estas porciones de tiempo que no pueden ser medidas con el TSC, debido a que en este estado de ahorro de energía, el TSC no estaría funcional.

Las arquitecturas de 32 bits no poseen implementados los notificadores para manejo de eventos idle, por lo tanto se puede probar de dos formas diferentes. O se genera una tarea que emule los ingresos a estados idles (idéntica a la provista por el kernel) o se modifica el kernel para que ya la tenga incorporada. Además en aquellas arquitecturas en las cuáles sus TSCs no se detienen también se puede probar dado que el driver se activa con aquellos estados en los cuáles su numero sea mayor a 2 (ósea en los cuáles sus TSCs se pueden detener). Pero también se lo puede configurar a 1 y posteriormente probar el reloj aunque se adelantará porque el manejo del idle le agregará al tiempo del sistema el tiempo en idle calculado.

6.3.2.3. Monoprocesador, manejo de idles profundos y cambios de frecuencia:

Existen en el kernel notificadores de inicio-fin de cambios de frecuencia gracias al subsistema cpufreq que se encarga de las funciones relacionadas al modulo de gestión de energía. Utilizando estos notificadores lo que se debe hacer es registrar una función interesada en tales eventos, que luego del cambio de frecuencia configurará el reloj de manera que exprese sus cálculos de acuerdo al cambio de frecuencia; en detalle:

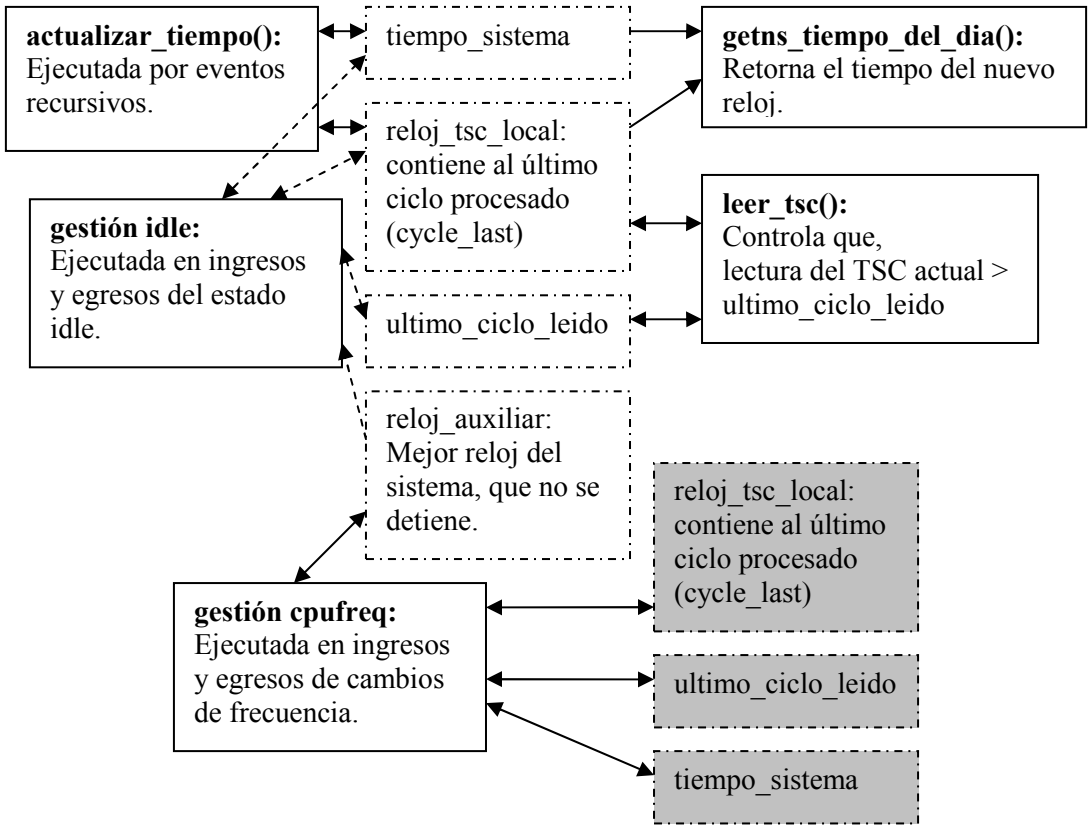
Entrada cambio de frecuencia:

- Actualización del tiempo del sistema.

Salida cambio de frecuencia:

- Se consulta al driver que maneja los cambios de frecuencia si conoce cuanto tiempo tardo en realizar el cambio, si lo sabe entonces agregar este tiempo al tiempo del sistema
- Dado que se cambio de frecuencia cierta cantidad de ciclos no van a tener la misma equivalencia de nanosegundos que antes se tenia, por lo que hay que recalcular al multiplicador del reloj y actualizarlo.

Esquema conceptual de funcionamiento:



Esta solución maneja cualquier política de cambio de frecuencia implementada por el subsistema cpufreq: ondemans, conservative, userspace, performance, powersave, etc. Y es adecuada para aplicaciones que posean requerimientos precisos del tiempo, y además

pretendan un crecimiento monotónico del tiempo, esta solución es similar a la anterior pero mejora el manejo del tiempo en estados de cambio de frecuencia.

6.3.2.4. Multicore, manejo de idles profundos:

Tengamos en cuenta que ahora existe un thread idle por CPU y que siempre la notificación se realiza desde la CPU que va a entrar en estado idle C3; ósea una CPU no nos va a notificar que otra CPU esta entrando en estado idle C3. En este caso se va a agregar una variable de "ciclos_en_idle" (guarda la cantidad de ciclos cuando el TSC se detuvo, desde el inicio del sistema) e “ini_idle” (guarda el tiempo de inicio cuando el TSC se detuvo) para cada CPU, en detalle:

En el núcleo general se modifica la función leer_tsc(). Cuando se lean los ciclos del actual procesador se deberá sumar a esta cantidad el valor de la variable "ciclos_en_idle" de la CPU que lee y luego seguir con la comparación del último ciclo leído (como control de monotonicidad).

Se tienen dos casos:

-Cuando la CPU que ingresa en estado idle es diferente a la que maneja la actualización del tiempo:

Entrada idle:

- Tomar el tiempo de inicio desde el mejor reloj de referencia.

Salida idle:

- Tomar el tiempo final y calcular el intervalo de tiempo en nanosegundos en el que se estuvo detenido, pasarlo a ciclo de reloj y actualizar la variable correspondiente a su CPU "ciclos_en_idle".

-Cuando la CPU que ingresa en estado idle es también la que maneja la actualización del tiempo:

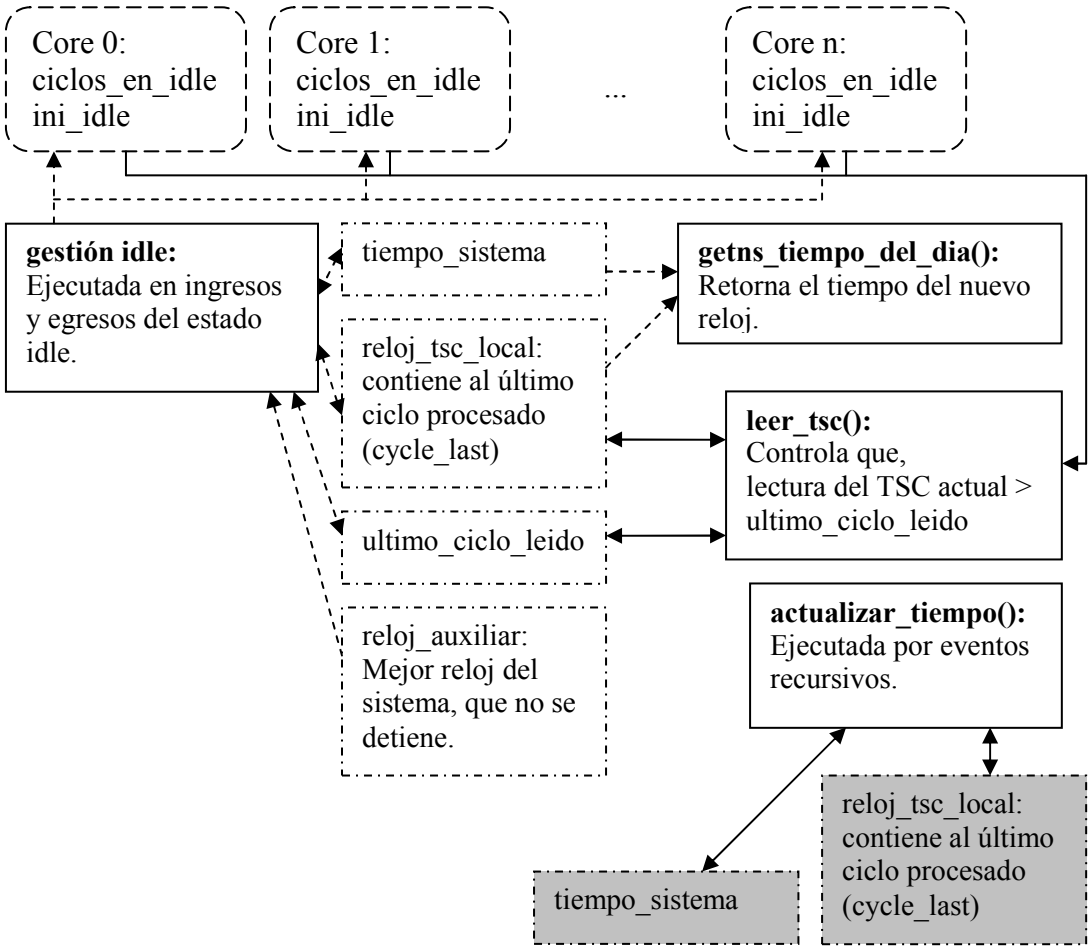
Entrada idle:

- Actualizar el tiempo del sistema y para todas las demás CPUs que estén en estado idle, resetearle la variable de inicio del idle al actualmente leído tiempo de inicio de estado idle del procesador que maneja las actualizaciones del tiempo.

Salida idle:

- Calcular el tiempo en el cual se estuvo en idle, finalmente se obtendrá su equivalente en ciclos de este tiempo en idle y se actualizara su variable "ciclos_en_idle".

Esquema conceptual de funcionamiento:



6.3.2.5. Multicore, manejo de idles profundos y cambios de frecuencia: Solución A.

Esta solución es compleja; por dos razones importantes que afectan la performance de la solución. Cómo manejar los cambios de frecuencia, si cada procesador puede cambiar de frecuencia; por lo que sus ciclos_en_idle tendrán una representación en nanosegundos diferentes según el multiplicador actual. ¿Habría que recalcular ciclos_en_idle en cada cambio de frecuencia?. La variable ciclos_en_idle intenta que cada vez que se soliciten los ciclos de la CPU actual retorne los ciclos que serian "como si no se hubiera detenido el TSC". Con lo que independientemente del procesador que lea este TSC semi-modificado con una variable de corrección, retornaría un valor que simula un TSC que no se detiene en estado idle profundo. Ósea intenta tener un TSC corregido que siempre corre y nunca se detiene. Una alternativa seria guardar no ciclos sino nanosegundos en estado idle profundo pero se mejorará generando en verdad un TSC virtual el cual funcionará como una especie de TSC corregido ante cambios de frecuencia e idles profundos.

Para manejar los cambios de frecuencia se generará una estructura que albergará un tsc_virtual, una por cada CPU. Cada vez que se intente leer los ciclos del procesador

actual se leerá el valor de esta estructura la cual se actualizará cada vez que se lea del TSC, se ingrese a estados idles profundos y también cada vez que se cambie de frecuencia, en detalle:

En general todas las arquitecturas se inician con TSC sincronizados, luego se desincronizan cuando se ingresa a estados idle profundos y cuando cambian de frecuencia.

Estructuras y variables principales:

- Se va a incorporar una estructura que mantiene el tiempo inicial del idle profundo (ini_idle), el tiempo inicial del cambio de frecuencia (ini_freq) y un filtro para solo atender a los eventos idle profundos, llamado tiempo_inicial. Habrá una estructura para cada procesador.
- Se va a incorporar una estructura que mantiene la frecuencia actual (freq) del procesador actual y el multiplicador (mult), llamado fremu_virtual, para cada procesador.
- Se va a incorporar una estructura que mantiene el tsc_virtual, que consta del ultimo_ciclo_leido en el procesador y el valor corregido, para cada procesador.
- Por razones de performance, incluso el procesador que maneja las actualizaciones del tiempo posee la estructura tsc_virtual. Así se evitan en lo posible los locks producidos por estar leyendo la estructura de otro procesador y la cual se puede modificar. Porque si se lee del procesador que actualiza el tiempo, hay que tener presente que éste mismo puede también cambiar de frecuencia. Y en lugar de hacer un rebase de frecuencia para todos los procesadores con respecto al procesador manejador de las actualizaciones del tiempo porque modifico su multiplicador y frecuencia es mejor crear una estructura externa. Esta estructura externa contendrá los valores de frecuencia de referencia y multiplicador de referencia para todos los tsc_virtuales, la cual será de solo lectura y será inicializada con datos de la primer CPU calibrada en el inicio. Ésta estructura de fremu_virtual de referencia inicial, será llamada fremu_referencia.
- Se cambiará la semántica del significado del clocksource_tsc_local; el cual será igual pero el campo cycle_last se referirá al último ciclo leído en general (producto de una lectura sobre el tsc_virtual de cualquier procesador) procesado. El cycle_last se utiliza para controlar las actualizaciones del tiempo del sistema. El mult si bien se inicializa y será igual al mult del fremu_referencia éste no se utiliza; pero se lo deja para poder ser utilizado en un futuro como mult con ajuste. La estructura nueva se llamará clocksource_tsc_global y se cambiarán algunos nombres de las funciones de ayuda.
- Se definirá un lock para cada procesador, que resguardará los cambios en los TSCs virtuales.
- Se definirá un lock para cada procesador, que resguardará los cambios en los fremu_virtuales.
- Se definirá un lock para la actualización del tiempo del sistema.
- Se definirá una variable del último valor en ciclos del tsc_virtual leído en general, llamado ultimo_general. Se utiliza para guardar el ultimo ciclo leído

general y mayor, siempre va a ser mayor o igual al que el campo cycle_last del clocksource_tsc_global.
- Se definirá un lock para la actualización del último ciclo leído en general del tsc_virtual.

Esta solución puede ser implementada de las siguientes maneras. Si se va a modificar el kernel, en el main, setear la estructura: se puede reutilizar la frecuencia detectada por el inicio del TSC (antes de ser marcada como inestable, en caso que lo fuera) o recalcularla. Una vez hallada la frecuencia se procederá a calcular el multiplicador; con estos datos se puede o bien setear todas las estructuras de los procesadores o bien recalcularla para cada procesador. En caso de ingresarla como modulo se puede recalcular la frecuencia para cada procesador y opcionalmente se puede definir una tarea cíclica que recorra de a dos CPUs (la CPU que maneja la actualización del reloj contra cada una de las demas). Con esto último se puede hallar un valor de desvío que definirá un valor de corrección que será agregado al tsc_virtual cada vez que se lea de él. Éste valor puede respetar un umbral de posible margen de error en cantidad de ciclos iniciales, actuará como una especie de sincronización de valores del TSC virtual inicial. En nuestro caso optamos por cargar el modulo en tiempo de ejecución y además una vez calculada la frecuencia, ésta será utilizada para la configuración de la información de todos los procesadores. De esta forma evitamos los recálculos para cada procesador para que detecten su frecuencia de funcionamiento. Aunque tenemos una condición más. Antes de cargar el modulo se debe asegurar de no tener una gestión de frecuencia automática, para mejorar la sincronización virtual del valor inicial del tsc_virtual debido a que por default el driver cpufreq configura el governor ondemand. También se puede modificar la configuración a userspace y se recompila el kernel o bien se configura manualmente para cada procesador de la siguiente forma:

Ej: procesador 0:

-configurar el governor a modo usuario:

#cd /sys/devices/system/cpu/cpu0/cpufreq
#echo userspace > scaling_governor

-ver las frecuencias disponibles:

#cat scaling_available_frequencies

-seleccionar una frecuencia y setearla:

#echo “frecuencia_base_selecionada” > scaling_cur_freq

-Repetir los pasos para todos los procesadores disponibles.

La estructura que mantiene el tsc_virtual por cada procesador es inicializada con el ciclo leído luego de la calibración del procesador inicial/procesadores. En el caso de modificación del kernel todas tendrán el mismo valor, porque no se inicio el cpufreq para gestión de frecuencia, ni se entro en estado idle profundo. En nuestro caso se va a generar threads para cada procesador que se sincronicen antes de leer los ciclos y actualicen su ultimo_ciclo_leido.

Manejo de idles profundos:

Refactorizar el manejo estado idle profundo: el manejo será similar al anterior salvo que no se mantendrá en una variable separada el tiempo que se estuvo en estado idle, sino que ésta será procesada y agregada al tsc_virtual de la siguiente manera:

Entrada idle:

- Cuando una CPU cualquiera ingrese en estado idle se verifica si no es un idle "significativo" si no lo es entonces se actualizará el reloj tsc_virtual. Se obtiene la diferencia entre el último ciclo leído en ese procesador y el actual del mismo procesador, calcula los nanosegundos según la frecuencia actual. Luego toma esos nanosegundos y calcula su equivalencia en ciclos del procesador virtual de referencia, fremu_referencia (usa el mult) con el shift del reloj definido. Posteriormente agrega éste valor al valor del tsc_virtual y además actualiza el último ciclo leído al actual. Finalmente toma el tiempo de inicio desde el mejor reloj de referencia.

Salida idle:

- Verifica si la notificación de salida del idle no tiene una correspondiente entrada a idle en ese caso salir. Esto pasa cuando se produce un salir idle "significativo" por parte del manejador de interrupciones, basta con verificar si se dispone de un tiempo inicial de idle. En caso contrario tomar el fin del tiempo idle y calcular el intervalo de tiempo en nanosegundos en el que se estuvo detenido. Pasar estos nanosegundos a ciclos del procesador virtual de referencia, fremu_referencia (usa el mult) y el shift del reloj definido, actualiza el campo correspondiente al valor del tsc_virtual. Finalmente resetea el valor del tiempo inicial de idle para esta CPU.

Manejo de cambios de frecuencia:

Entrada cambio de frecuencia:

- Si la frecuencia nueva de funcionamiento es mayor a la actual entonces realizar una sincronización general de todos los cores.
- Se toma el tiempo inicial del cambio de frecuencia usando el mejor reloj de referencia.

Salida cambio de frecuencia:

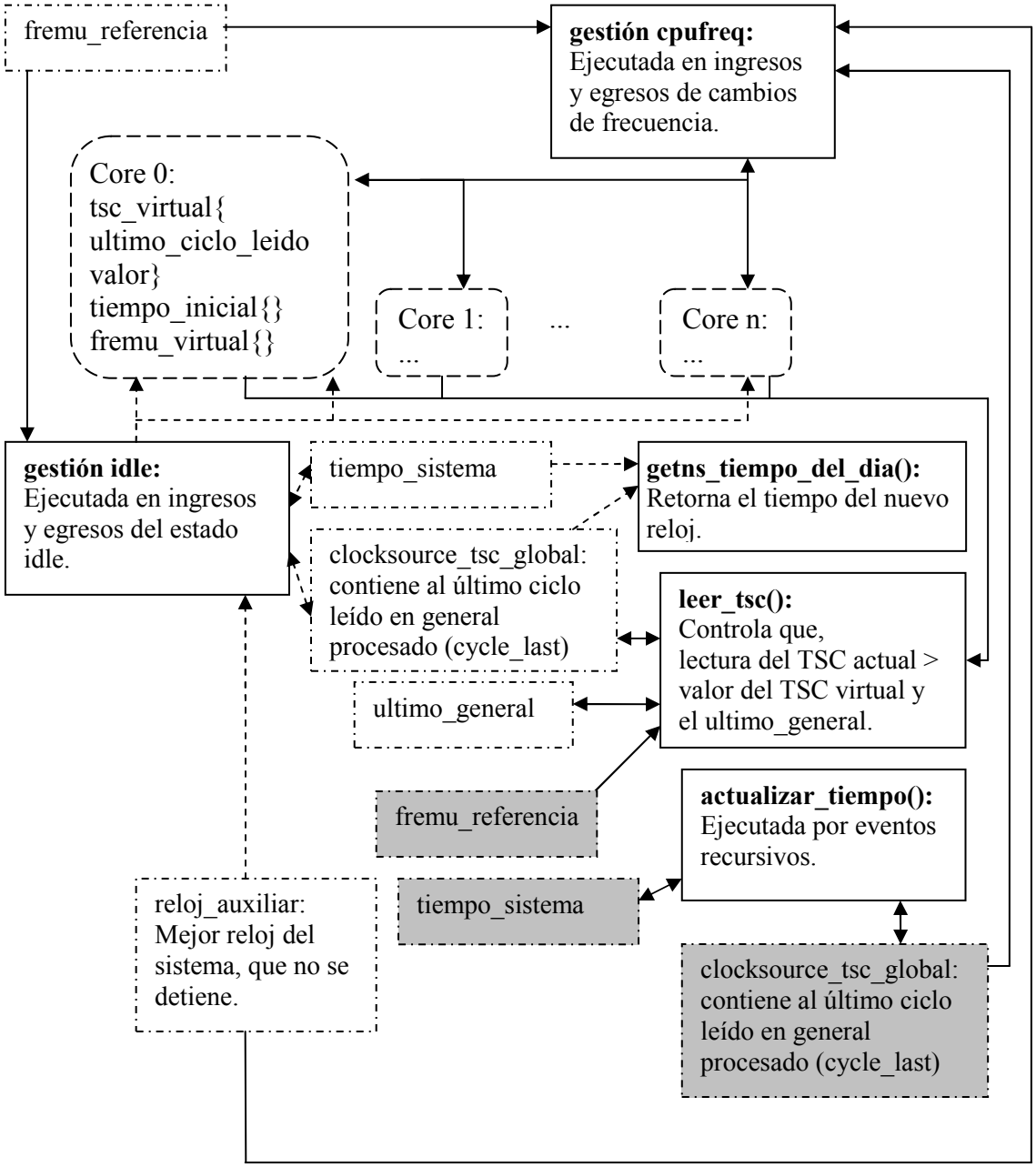
- Se toma el tiempo final del cambio de frecuencia entonces calcular este tiempo de nanosegundos a cuantos ciclos equivalen usando el fremu_referencia (mult) y el shift del reloj y agregarlo al valor del tsc_virtual. Luego actualizar el ultimo ciclo leído del procesador actual al valor actual del ciclo del procesador (se hace como una medida de reinicio). Dado que se cambio de frecuencia cierta cantidad de ciclos no van a tener la misma equivalencia de nanosegundos que antes se tenia, por lo que hay que actualizar el multiplicador del reloj, y la frecuencia nueva de funcionamiento.

En el núcleo:

- Cada vez que se solicite leer el TSC del procesador actual para calcular el tiempo se hará lo siguiente: Calcular la diferencia entre el último ciclo leído y el actual del procesador actual. Pasarlo a nanosegundos (usando el reloj y el multiplicador de la frecuencia actual de funcionamiento). Calcular con este valor la cantidad de ciclos que representarían según la escala de referencia (usando mult de fremu_referencia y el shift del reloj). Actualizar el valor del tsc_virtual, y por ultimo comparar este valor con la variable ultimo_general (control monotónico).

Esta solución maneja solo la política ondemand, debido a que esta política autogestiona los cambios de frecuencia y los cambios son producidos desde el mismo procesador que quiere realizar el cambio de frecuencia. En caso de los otros mecanismos de cambios de frecuencia, los cambios de frecuencia pueden ser iniciados por un procesador distinto al que cambia de frecuencia, por lo que esta solución no es aplicable en estos casos.

Esquema conceptual de funcionamiento:



6.3.2.6. Multicore, manejo de idles profundos y cambios de frecuencia: Solución B Final.

Ahora extenderemos la solución, para lograr una integración completa con el subsistema cpu_freq más un mecanismo extra de sincronización con un reloj del sistema.

Manejo de idles profundos:

Similar al anterior.

Entrada idle:

- Cuando una CPU cualquiera ingrese en estado idle se verifica si no es un idle “significativo” si no lo es entonces se actualizará el reloj tsc_virtual. Se obtiene la diferencia entre el último ciclo leído en ese procesador y el actual del mismo procesador, calcula los nanosegundos según la frecuencia actual. Luego toma esos nanosegundos y calcula su equivalencia en ciclos del procesador virtual de referencia, fremu_referencia (usa el mult) con el shift del reloj definido. Posteriormente agrega este valor al valor del tsc_virtual y además actualiza el último ciclo leído al actual. Finalmente toma el tiempo de inicio desde el mejor reloj de referencia.

Salida idle:

- Verifica si la notificación de salida del idle no tiene una correspondiente entrada a idle en ese caso salir. Esto pasa cuando se produce un salir idle "significativo" por parte del manejador de interrupciones, basta con verificar si se dispone de un tiempo inicial de idle. En caso contrario tomar el fin del tiempo idle y calcular el intervalo de tiempo en nanosegundos en el que se estuvo detenido. Pasar estos nanosegundos a ciclos del procesador virtual de referencia, fremu_referencia (usa el mult) y el shift del reloj definido, actualiza el campo correspondiente al valor del tsc_virtual. Finalmente resetea el valor del tiempo inicial de idle para esta CPU.

Manejo de cambios de frecuencia:

Tengamos en cuenta que ahora las notificaciones de cambio de frecuencia no siempre se manejan desde el procesador que quiere cambiar de frecuencia; ósea una CPU nos puede notificar que otra CPU esta entrando en un cambio de frecuencia. Aunque por lo menos al igual que los manejo del idle sabemos que la entrada y salida de cambios de frecuencia se va a realizar sobre el mismo procesador que maneja la notificación.

Entrada cambio de frecuencia: si la CPU que quiere cambiar de frecuencia es la actual o una diferente entonces el procedimiento es similar al de la solución anterior.

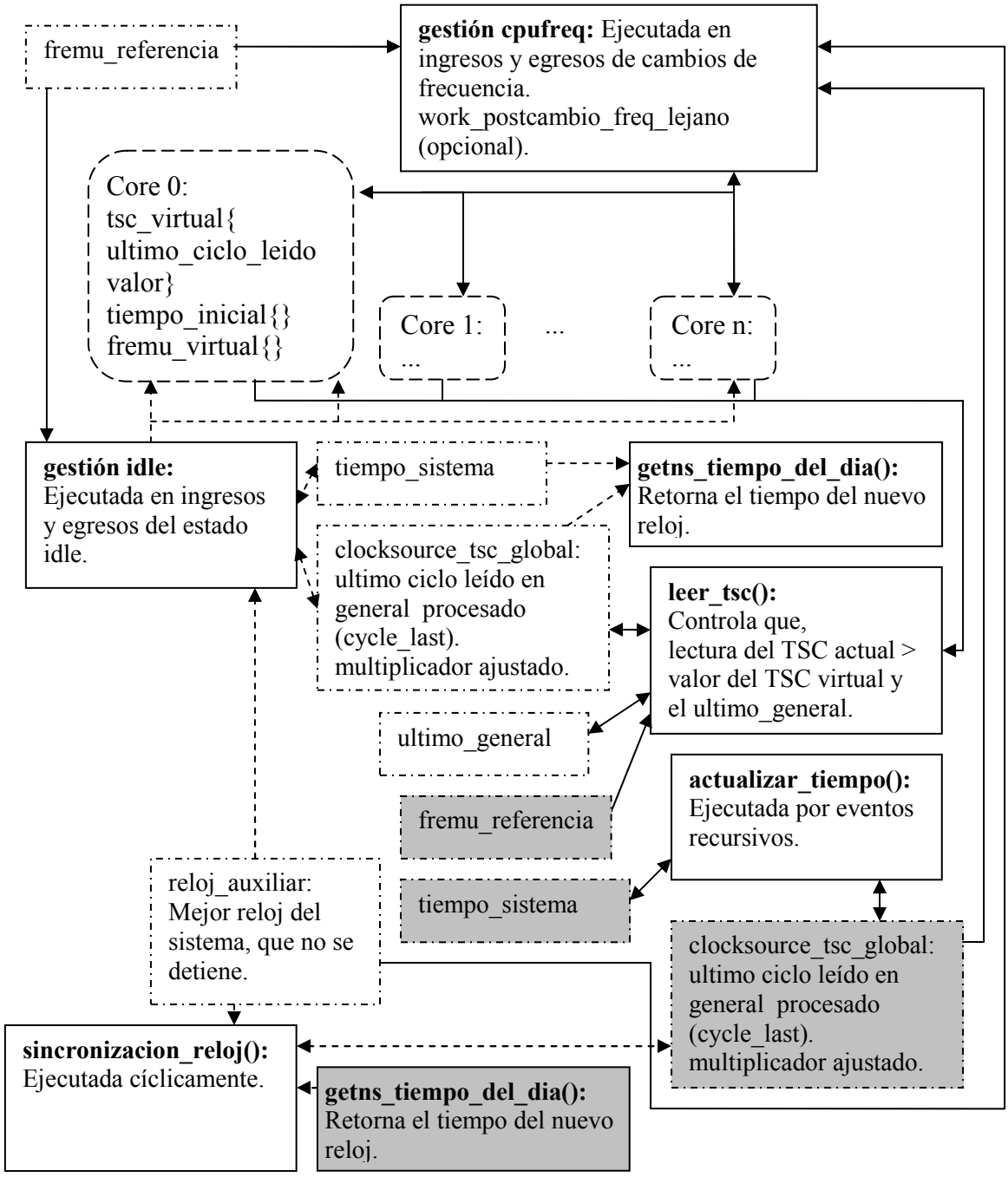
- Se toma el tiempo inicial del cambio de frecuencia usando el mejor reloj de referencia.

Salida cambio de frecuencia: si la CPU que termino de cambiar de frecuencia es la actual entonces el procedimiento es similar al de la solución anterior, sino se toma el tiempo final del cambio de frecuencia usando el mejor reloj de referencia; luego:

- Se genera una tarea la cual calcula el intervalo de tiempo de nanosegundos a cuantos ciclos equivalen usando el fremu_referencia (mult) y el shift del reloj y lo agrega al valor del tsc_virtual. Luego actualiza el ultimo ciclo leído del procesador actual al valor actual del ciclo del procesador (se hace como una medida de reinicio). La sincronización entre la CPU manejador y la CPU objetivo del cambio de frecuencia se hace mediante una estructura de sincronización. Debido a que se cambio de frecuencia cierta cantidad de ciclos no van a tener la misma equivalencia de nanosegundos que antes se tenia, por lo que hay que actualizar el multiplicador del reloj, y la frecuencia nueva de funcionamiento.

Para que el nuevo reloj se mantenga sincronizado con un reloj de referencia se realiza una sincronización a nivel de reloj en cada marca. Esta sincronización a nivel de reloj se efectúa de la siguiente manera, en cada marca se ejecuta un thread cuyo trabajo es observar la hora del reloj de referencia y observar la hora del nuevo reloj. Dependiendo de si el nuevo reloj esta adelantado o retrasado, el thread modifica ciertos parámetros del nuevo reloj (campo mult del reloj clocksource_tsc_global) para realizar un ajuste de incrementos por marca. Lo que hace es modificar el multiplicador para que la cantidad de tiempo que represente por ciclo del TSC sea mayor o menor dependiendo de la necesidad del ajuste. Este mecanismo ira ajustando al reloj para mantenerlo lo más sincronizado posible con respecto al reloj de referencia. Como resultado se obtiene una diferencia de desincronización promedio de cómo máximo +/- 210ns.

Esquema conceptual de funcionamiento:



7. PRUEBAS DE FUNCIONAMIENTO

En este capitulo presentaremos las pruebas realizadas sobre algunas de las soluciones vistas en el capitulo anterior. Para ver más detalles ver el ANEXO F.

7.1. Monoprocesador-Multicore e independiente de cambios de frecuencia e idles profundos

Ver Anexo F.1 para más detalles.

7.1.1. Prueba de funcionamiento general

Se disponen de dos threads que leen la hora continuamente.

Ejecución de los procesos de prueba:

Se ejecutan dos threads, en un core cada uno, que verifican constantemente si existe una desincronización en cada lectura del tiempo utilizando para ello una variable con la cual podrán comparar el tiempo leído actualmente contra el ultimo leído por los threads. También posee un thread de observación de tiempo de ejecución para controlar el fin del análisis. Al final se imprimirá la cantidad total de verificaciones, cantidad total de desincronizaciones y el porcentaje total de fallas (con respecto a lecturas del tiempo no sincronizadas). El tiempo total de análisis de los threads será de 40 segundos.

Resultado: En donde dos threads de análisis leen la hora continuamente, sin espera.

```
/*Cantidad de lecturas de la hora que se pudieron hacer durante el
*análisis*/
Cantidad total de verificaciones = 205890324

/*Cantidad de desincronizaciones que detecto el thread de análisis,
*deberá ser siempre 0*/
Cantidad total de desincronizaciones según la prueba = 0

/*Cantidad total de lecturas de TSC y lecturas TSC desincronizadas*/
Total lecturas TSC: 205906495, desincronizadas: 2216759

/*Porcentaje de lecturas de TSC desincronizadas corregidas*/
Porcentaje total de lecturas corregidas = 1.07658%(aprox.)

/*Cantidad de reverificaciones en getns_tiempo_del_dia() este valor
*indicara la cantidad de correcciones de la ultima hora leída
*corregida*/
Horas corregidas: 0

/*Umbral máximo de desincronización entre lecturas de TSC*/
Máxima desincronización de ciclos: 3220306

/*Umbral máximo de correcciones de la ultima hora leída*/
Máxima corrección de hora: 0 ns
```

Finalmente la hora siempre se mantiene creciente con el nuevo reloj.

7.2. Monoprocesador con manejo de idles profundos

Ver Anexo F.2 para más detalles.

7.2.1. Pruebas de funcionamiento generales

Prueba:

- Se va a cargar el driver del nuevo reloj que contiene al reloj y sus funciones.
- Si se opta por no modificar el kernel entonces también se debe cargar el modulo que emule los ingresos a estados idle.
- Se va a lanzar un thread que continuamente lea la hora y en cada lectura verifica si existe desincronización de lectura de reloj.

Ejecución de los procesos de prueba:

La simulación de eventos de detención del TSC es producida por un thread que llama a cpu_idle() cada 1000 ms durante 20 segundos (simulando llamadas a idles profundos). Como en el ambiente de prueba el TSC no se detiene entonces se configurará la sensibilidad del manejador a 1 (la sensibilidad es utilizada para conocer el estado actual de la CPU). Esto ultimo se hace para probar la funcionalidad, aunque el TSC siga corriendo; en este caso se debería modificar el manejo del fin del idle para que reconfigure el ultimo ciclo leído al actualmente leído porque el TSC sigue corriendo.
Existe un thread en un core que verifica constantemente si existe una desincronización en cada lectura del tiempo utilizando para ello una variable con la cual podrá comparar el tiempo leído actualmente contra el ultimo leído por el thread.
También se dispone de un thread de observación de tiempo de ejecución para controlar el fin del análisis. Al final se imprimirá la cantidad total de verificaciones, cantidad total de desincronizaciones y el porcentaje total de fallas (con respecto a lecturas del tiempo no sincronizadas). El tiempo total de análisis del thread será de 40 segundos.

Resultado: En donde el thread de análisis lee la hora cada 500 microsegundos.

```
Cantidad total de verificaciones = 55814
Cantidad total de desincronizaciones según la prueba = 0.

Total lecturas TSC: 63983, desincronizadas: 0
Porcentaje total de lecturas corregidas = 0%(aprox.)
Horas corregidas: 0
Máxima desincronización de ciclos: 0
Máxima corrección de hora: 0 ns
```

Resultado: En donde el thread de análisis lee la hora continuamente, sin espera.

```
Cantidad total de verificaciones = 219331418
Cantidad total de desincronizaciones según la prueba = 0.

Total lecturas TSC: 219342993, desincronizadas: 21465
Porcentaje total de lecturas corregidas = 0.00978%(aprox.)
Horas corregidas: 0
Máxima desincronización de ciclos: 35784454
```

```
Máxima corrección de hora: 0 ns
```

Como se esperaba las lecturas del tiempo del nuevo reloj están sincronizadas con 0% de fallas en lecturas continuas de la última hora del nuevo reloj.

7.3. Multicore, manejo de idles profundos y cambios de frecuencia. Solución Final.

Ver Anexo F.3 y Anexo G para más detalles.

7.3.1. Prueba de reloj

Prueba:

- Se va a cargar el driver del nuevo reloj que contiene al reloj y sus funciones.
- En nuestro caso optamos por no modificar al kernel por lo que también se debe cargar el modulo que emule los ingresos a estados idle (cada 10 segundos se emitirá el evento de ingreso a idle profundo).
- Se van a lanzar dos threads que continuamente lean la hora y en cada lectura verificarán si existe desincronización de lectura de reloj.
- Se van a realizar cambios automáticos de frecuencia cada 1 minuto.
- Se van a ejecutar todas las tareas durante dos horas.

Resultado:

En donde los threads de análisis leen la hora continuamente, sin espera. En la consola se iran imprimiendo los cambios de frecuencia por minuto. Dando como resultado lo siguiente:

```
Cantidad total de verificaciones = 12011623580
Cantidad total de desincronizaciones según la prueba = 0.

Total lecturas TSC: 12014155956, desincronizadas: 3266775
Porcentaje total de lecturas corregidas = 0.02719%(aprox.)
Máxima desincronización de ciclos: 19322033
Máxima corrección de ciclos: 6583461 ns
Horas corregidas: 0
Máxima corrección de hora: 0 ns
```

Como podemos observar la hora siempre se mantiene sincronizada ante cambios de frecuencia y manejo de idles profundos, utilizando como reloj al TSC.

7.3.2. Prueba de sincronización a nivel de reloj

En la siguiente prueba se leerá la hora secuencialmente, un thread para el nuevo reloj y otro para el reloj del sistema, probando el mecanismo de sincronización cada 1ms, durante 20 minutos. Si bien los dos threads se sincronizan para iniciar, posteriormente se lee la hora secuencialmente y no se produce una sincronización entre threads, para que se inicie la lectura del tiempo de cada reloj, para evitar afectar en lo posible al mecanismo del cambio de frecuencia, esto producirá el incremento de la diferencia.

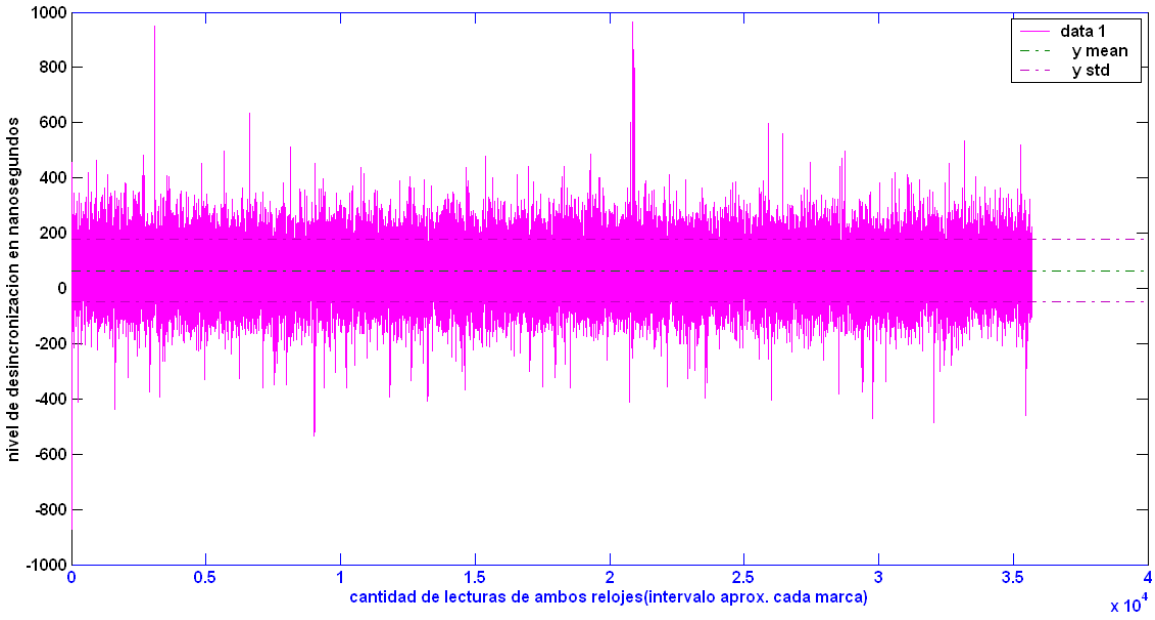
Figura 12: Resultados prueba de sincronización de reloj

Minuto	Reloj del sistema en segundos.nanosegundos	Nuevo Reloj en segundos.nanosegundos	Diferencia en nanosegundos
1	1352762196.840142857	1352762196.840143033	176
2	1352762256.840153473	1352762256.840153537	64
3	1352762316.840154101	1352762316.840154125	24
4	1352762376.840079441	1352762376.840079616	175
5	1352762436.840165904	1352762436.840165974	70
6	1352762496.840156895	1352762496.840156904	9
7	1352762556.840123371	1352762556.840123349	-22
8	1352762616.848157104	1352762616.848157059	-45
9	1352762676.848154939	1352762676.848155074	135
10	1352762736.848156057	1352762736.848156264	207
11	1352762796.848156685	1352762796.848156785	100
12	1352762856.848156546	1352762856.848156598	52
13	1352762916.848148374	1352762916.848148444	70
14	1352762976.848151447	1352762976.848151593	146
15	1352763036.848151796	1352763036.848151709	-87
16	1352763096.848154101	1352763096.848154121	20
17	1352763156.848154520	1352763156.848154637	117
18	1352763216.848153612	1352763216.848153659	47
19	1352763276.876157314	1352763276.876157266	-48
20	1352763336.876153193	1352763336.876153255	62

Como resultado se obtiene una diferencia de desincronización de cómo máximo +/- 210ns.

En la siguiente figura, el eje “x” representa la cantidad de lecturas de reloj realizadas en ambos relojes (el reloj del sistema y el nuevo reloj); y el eje “y” representa la diferencia en nanosegundos de ambas lecturas. Como se puede ver, la diferencia en general esta en el rango de -200 y +210 ns.

Figura 13: Grafico parcial de resultados de la sincronización durante 20 minutos, prueba desde espacio del kernel.



7.4. Beneficios del Nuevo Reloj

Mediante las diferentes pruebas se detecto que cualquier reloj del sistema es afectado por el mecanismo de gestión de energía (no solo el que trabaja con el TSC) desde el punto de vista del rendimiento del manejador del reloj respectivo. Por ejemplo un reloj puede ser más efectivo con un determinado mecanismo de gestión de energía pero menos efectivo con otro mecanismo de gestión de energía. En la actualidad el mejor reloj del sistema seleccionado por el kernel de Linux en un ambiente desincronizado es el HPET.

El nuevo reloj realiza sincronizaciones parciales en cada lectura disminuyendo la cantidad de medida de corrección sobre el TSC virtual (cómo el nuevo reloj se integra al mecanismo de cambios de frecuencia en varios ambientes se lo describe en el Anexo G). En un ambiente normal de uso de los cores el nuevo reloj provee un beneficio del 56% en velocidad de ejecución y una mayor precisión con respecto al mejor reloj del sistema.

Diferencias entre el nuevo reloj y la característica constant_tsc para obtener un TSC constante:

Nuevo reloj	**Característica constant_tsc**
Soporte para ambientes desincronizados.	No provee soporte para ambientes desincronizados.
Integración al modulo de gestión de energía.	Corre siempre a máxima frecuencia. No provee integración al modulo de gestión de energía.

Diferencias con respecto al comportamiento del kernel actual en un ambiente común desincronizado:

Nuevo reloj	**Reloj del kernel actual**
Mejora en un 56% al mejor reloj del sistema HPET	Mejor reloj HPET
Precisión TSC	Precisión HPET
Requiere un mínimo de proceso extra por lectura del TSC.	Se lee del reloj directamente sin proceso de verificación extra.
En arquitecturas x86 de 32 bits requiere que el soporte para estados idles profundos sea implementado como un parche del kernel si es que se requiere la máxima precisión en el tiempo. Aunque, se puede levemente modificar la solución, con solo una perdida en la precisión del tiempo del sistema.	No le afectan los estados idle.

7.5. Comportamiento del Nuevo Reloj

En los siguientes gráficos se mostrarán como es el comportamiento y forma del costo del mejor reloj del sistema vs el nuevo reloj en lecturas consecutivas por 1 minuto en dos ambientes extremos de gestión (con frecuencia ondemand y con frecuencia estable). Las graficas son parciales debido a la cantidad de los datos capturados.

En la siguiente grafica se superponen el costo en nanosegundos que necesita cada reloj para obtener la hora en un ambiente donde la frecuencia de la CPU es estable. En este caso se compara al mejor reloj del sistema HPET y al nuevo reloj. Como se puede observar, las lecturas sobre el nuevo reloj se ejecutan con mayor rapidez, lo que permite realizar una mayor cantidad de lecturas que las respectivas del HPET. Sobrepasa en un 13% en cantidad de lecturas. Lo cual se debe en gran medida a que las lecturas sobre el HPET son más costosas, debido a que el HPET es un dispositivo externo al microprocesador.

Figura 14: Gráfico de lecturas del HPET con frecuencia estable de CPU.

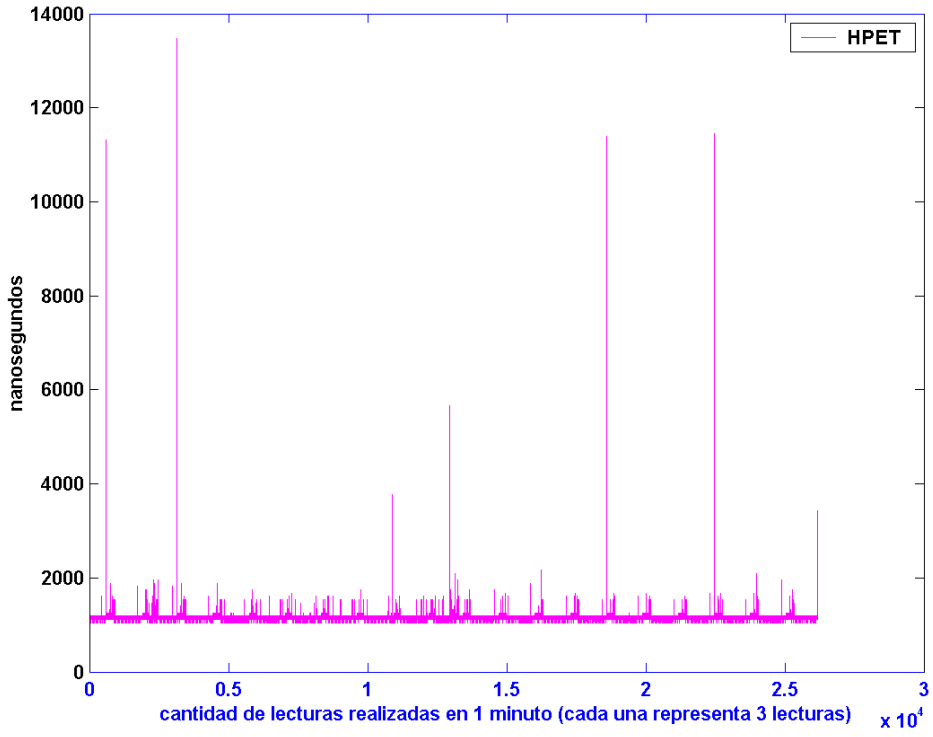
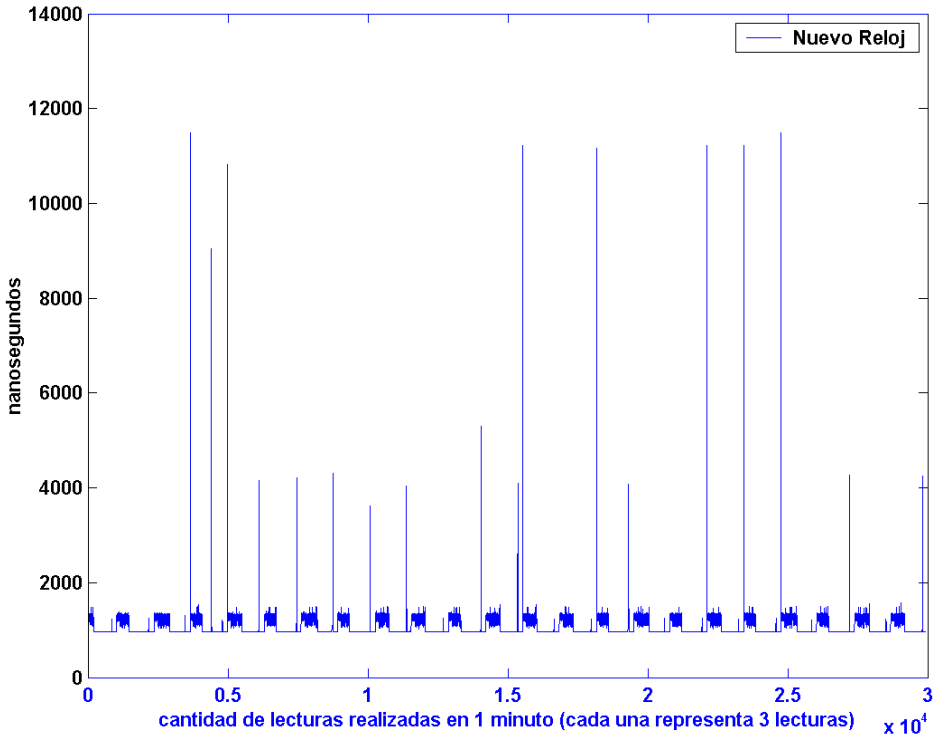
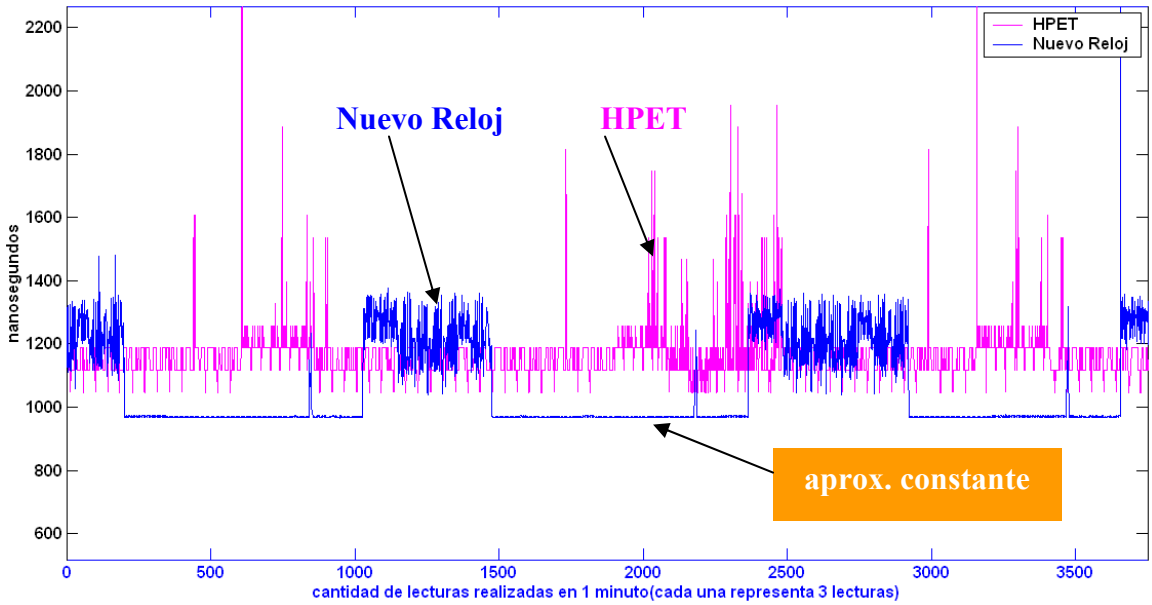


Figura 15: Gráfico de lecturas del nuevo reloj con frecuencia estable de CPU.



La siguiente gráfica mostrará un acercamiento de la graficas de lecturas del HPET y el nuevo reloj superpuestas en una sola gráfica. La cual indica que cada cierto intervalo de tiempo, el tiempo de obtener la hora con el nuevo reloj es casi constante.

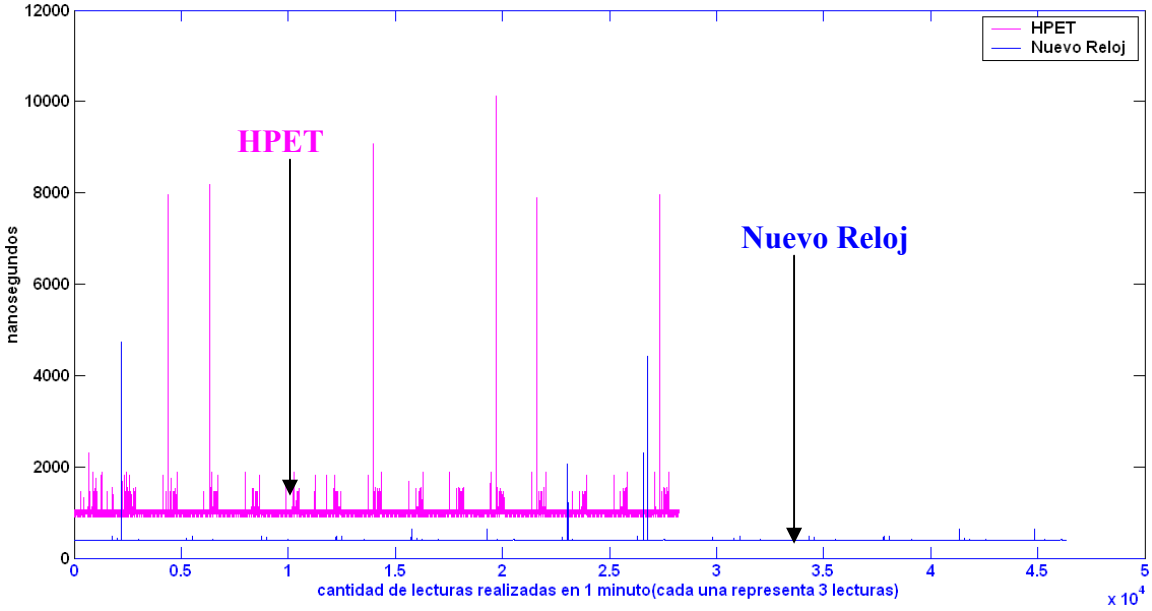
Figura 16: Gráfico parcial de lecturas del HPET y el nuevo reloj con frecuencia estable de CPU.



En la siguiente gráfica se superponen el costo en nanosegundos que necesita cada reloj para obtener la hora en un ambiente donde la frecuencia de la CPU es gestionada bajo el

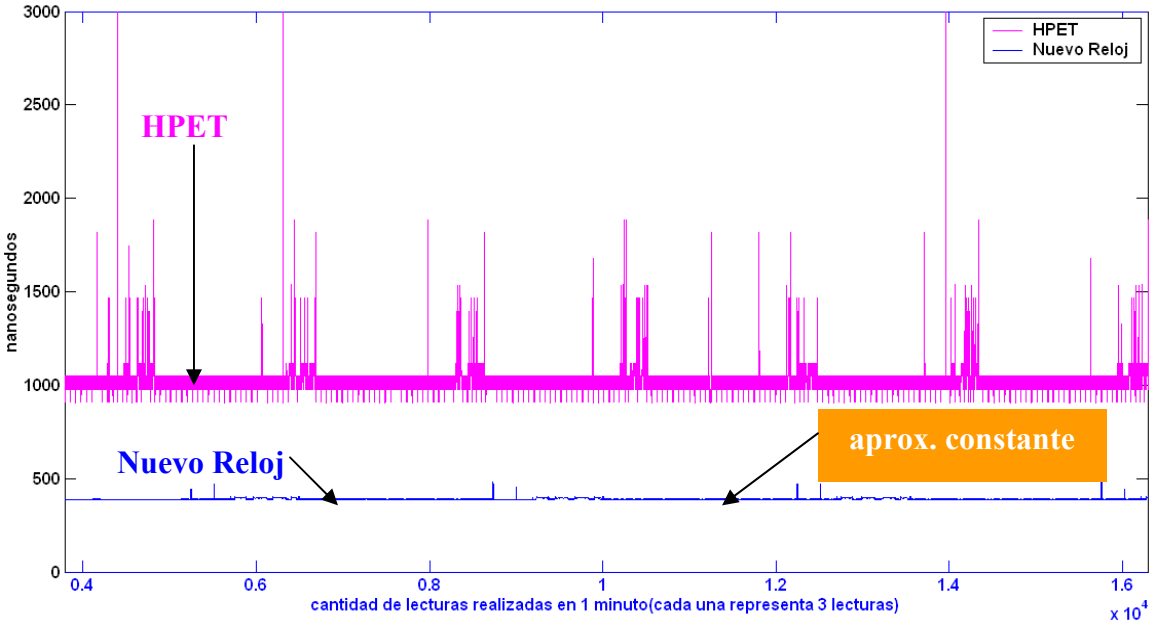
modo ondemand. Este modo indica que la CPU configurará su frecuencia dependiendo del estado de uso actual de la CPU. En este caso se compara al mejor reloj del sistema HPET y al nuevo reloj. Como se puede observar, las lecturas sobre el nuevo reloj se ejecutan con mayor rapidez lo que permite realizar una mayor cantidad de lecturas que las respectivas del HPET. En general el costo de ejecución del nuevo reloj no supera los 390 ns y sobrepasa en un 39% a la cantidad de lecturas del HPET.

Figura 17: Gráfico de lecturas de reloj con frecuencia ondemand de CPU (es el default del kernel).



La siguiente gráfica mostrará un acercamiento de la gráfica anterior. La cual indica que en general, el tiempo de obtener la hora con el nuevo reloj es casi constante.

Figura 18: Gráfico parcial de lecturas de reloj con frecuencia ondemand de CPU (es el default del kernel).



7.6. Uso del Nuevo Reloj

Los archivos fueron compilados con gcc-4.4 para 32 bits.

El reloj consta de los siguientes archivos:

-unicoReloj.c: contiene el driver del nuevo reloj (manejo de detenciones del TSC + cambios de frecuencia; reloj monotónico).
-unicoReloj.h: contiene las cabeceras del nuevo reloj.
-dormir.c: (opcional) es el archivo que en caso de las arquitecturas x86 de 64bits se puede agregar para el manejo de los cpuidle profundos como un modulo. En caso de no agregarla, no afecta a la monotonicidad del nuevo reloj, se agrega para que incorpore al TSC virtual el lapso de tiempo en el cual el TSC estuvo detenido, mejorando la precisión del tiempo. Notar que requiere que el driver que maneje estos eventos implemente el subsistema de manejos de idle genérico. En caso contrario, se deberá parchear el driver utilizado, incorporando una notificación en su código de entrada/salida de tales estados idle hacia la función ya implementada en dormir.c. En este caso, dormir.c esta modificado para las pruebas de simulaciones de idle, recibe como parámetro el tiempo durante el cual se emularan las llamadas a las notificaciones de idles, por default este es de 20 segundos. El intervalo de tiempo entre llamadas a idle es de 10 segundos.

El nuevo reloj es cargado en el sistema como un modulo, por ejemplo:

Resumen de ejecucion:

1- Ubicar los fuentes en una carpeta
2- Abrir un terminal e ingresar a dicha carpeta.
3- #make //generar el modulo .ko
4- #insmod nombrearchivo.ko //instala un modulo, y listo.
5- #dmesg //para ver el log
6- #rmmod nombremodulo //remueve un modulo
7- También se pueden consultar los módulos haciendo #lsmod

Uso 1:

Uso del reloj desde espacio de usuario:

Archivos utilizados:

-Makefile: es el makefile.
-unicoReloj.c: contiene el modulo del reloj a ser cargado.
-relojes.c: este archivo imprime la hora de todos los relojes definidos en el sistema.
-s: es el shell script para imprimir todos los relojes del sistema y también el nuevo reloj.

-copiar todos los archivos a una carpeta en el escritorio (o donde se quiera).
Primero se cargará el modulo, ejecutar en la línea de comandos:
#cd “ir a donde están las fuentes”
#make
#insmod unicoReloj.ko

En este punto ya esta cargado el modulo que contiene al reloj con lo que ya se puede obtener el tiempo del nuevo reloj de software.

#cat /proc/tiempo leerá la hora actual en segundos.nanosegundos

Para ver todos los relojes del sistema ejecutar:

#gcc –lrt –o relojes relojes.c
#./relojes

Se debe modificar el archivo del script apuntando al directorio donde se encuentran las fuentes (modificar la variable DIR2, ej: aquí se tienen los fuentes en el escritorio, DIR2=/home/ubuntu/Desktop).

Para ver todos los relojes y el nuevo reloj se debe ser root y ejecutar:

#sh s

Uso 2:

Uso del reloj desde otros módulos:

Basta con incluir el encabezado:
unicoReloj.h

y utilizar la llamada a la función:
getns_tiempo_del_dia(struct timespec *ts)

Aclaración final: El reloj por default se compara con el mejor reloj del sistema cada 1 ms. Posteriormente, realiza el ajuste de su multiplicador para tender a la sincronización entre el nuevo reloj y el mejor reloj del sistema.

8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones

Como hemos visto en este trabajo, abordar el problema de la sincronización del tiempo a nivel del TSC en el kernel Linux tiene sus dificultades. Principalmente por razones de poca bibliografía oficial y detallada de otras soluciones que abarquen al problema de la sincronización en arquitecturas "multicore", cosa que sucede con este tipo de software que esta en constante cambio. El hecho de que no contemos con una bibliografía oficial requiere de la realización de mucho análisis y pruebas a nivel del kernel. Debido a que el código en los sub-módulos del manejo del tiempo no están totalmente comentados y gran parte de ellos poseen una buena complejidad no deducible fácilmente.
Una de las contribuciones principales justamente esta referida a este último ítem. Al realizar un análisis en el capítulo 5 hemos documentado el esquema principal del manejo del tiempo en Linux de una manera abstracta, lo cual permitirá una correcta y fácil comprensión de los módulos afectados, transparentando su uso. Conocer al subsistema de manejo del tiempo es importante debido a que esto nos permite comprender como se comportará el reloj que se va a utilizar y cual será el reloj más adecuado para nuestras necesidades de aplicación.

Se analizaron sus diferentes partes, desde los dispositivos físicos que conforman los posibles orígenes de reloj a nivel físico, los sub-módulos del esquema de manejo del tiempo así como las relaciones entre estas. Luego se trazaron las funciones principales del subsistema de manejo del tiempo para una mejor comprensión.

Durante el análisis más detallado del esquema del tiempo en el capitulo 4, se explicaron los dos grandes cambios en el esquema principal. La incorporación de un subsistema para la administración de los orígenes de reloj y la incorporación de otro subsistema para la gestión de las fuentes de eventos del sistema. Tales subsistemas en general lo que buscaban era la eliminación de código duplicado en diferentes arquitecturas del kernel de Linux (no solamente para la arquitectura x86). Proveyendo un mejor manejo de los orígenes de reloj o fuentes de eventos de una manera flexible en tiempo de ejecución. Estos subsistemas definieron APIs para su uso en conjunto con estructuras base para el manejo del tiempo.

Los primeros cinco capítulos nos dan una visión global y detallada de todo el modulo de manejo del tiempo. Posteriormente se diseño una solución software para resolver el problema de la sincronización entre cores cuando el microprocesador no disponía de la característica constant_tsc, obteniéndose una hora sincronizada. Con lo cual, cada vez que se lee la hora utilizando el TSC local al proceso que lee, siempre se provee de una hora creciente y nunca decreciente, obteniéndose un reloj monotónico.

En el transcurso del diseño se procede a describir soluciones parciales y alternativas de solución, si bien éstas funcionan en determinadas arquitecturas de CPU, hay que tenerlas en cuenta cuando se utilizan en sus posibles ambientes de uso. Por ejemplo, tomemos el caso de una arquitectura multi-core donde el mecanismo de cambios de frecuencia nunca cambie y además sea ondemand. Si se pretende que la solución sea lo más eficiente posible en espacio y en tiempo de ejecución, la solución más adecuada es una solución específica porque la solución genérica es justamente genérica y van a

haber flujos de llamados a funciones con verificaciones que no se requieren en dicho contexto. Si bien no producen inconvenientes, siempre hay que tener en cuenta el contexto para adecuar la solución genérica y obtener la mayor eficiencia posible.

Posteriormente analizando los módulos de cambios de frecuencia y administración de energía se llegó al desarrollo de un reloj más fiel en cuanto a su representación del tiempo y completamente utilizable en cualquier arquitectura x86. Notemos que en las arquitecturas x86 de 32 bits, si se requiere de la mejor precisión posible para mantener al tiempo del sistema, se debe aplicar una leve modificación al kernel. Aunque la aplicación de dicha modificación no es obligatoria. En las arquitecturas x86 de 64 bits no es necesario modificar al kernel debido a que la misma ya esta actualizada hace muchos años para manejar completamente al subsistema de manejo de cambios de estados de CPU.

Luego, se realizaron pruebas sobre la solución final mostrando sus beneficios de eficiencia y resolución de nanosegundos. También se proveen de más pruebas de funcionamiento para las diferentes soluciones parciales, comportamiento y comparaciones del nuevo reloj, junto con otros resultados producto de realizar las pruebas sobre el nuevo reloj. Las mismas son explicadas en los Anexos F y G. Por ejemplo como afecta el mecanismo de gestión de energía en el comportamiento del nuevo reloj y en general.

Como conclusión final, el nuevo reloj posee menor sobrecarga que el uso del mejor reloj del sistema (mejora al HPET en un 56% en un ambiente de uso común) y mayor precisión (en nanosegundos). También, al estar integrado al mecanismo de gestión de energía permite al TSC cambiar de frecuencia. Incluso le permite al TSC detenerse sin necesidad de estar siempre corriendo a máxima frecuencia como en las arquitecturas de CPU con característica constant_tsc, justamente permite salvar energía. El uso eficiente de la energía es sumamente importante para cualquier arquitectura Linux.

En resumen se analizaron los siguientes puntos:

- Relojes hardware del sistema.
- Representación del origen de reloj para la interpolación de tiempo.
- Representación de las fuentes de reloj del sistema.
- Obtención de la hora actual del día.
- Mantenimiento de la hora del día.
- Trazado de funciones principales.
- Diseño de una escala de manejadores para la sincronización entre cores sin característica constant_tsc.
- Mejora del nuevo reloj desarrollado.
- Pruebas de funcionamiento.
- Descripción de beneficios del nuevo reloj.

8.2. Trabajo futuro

Dado que se ha descripto a todo el subsistema de manejo del tiempo de Linux, así como sus funciones genéricas para diferentes arquitecturas, se podrían realizar las siguientes actividades:

Integrar la solución planteada a la rama principal del kernel:
En el esquema de manejo del tiempo de Linux, la selección del mejor reloj del sistema asume que el reloj es único (cosa que no pasa con el TSC), lo cual produce inconvenientes en todo el subsistema de manejo del tiempo. Se podría modificar el subsistema para el manejo del tiempo genérico adecuando la solución a un nivel superior de generalización o agregando un reloj POSIX. Analizando incluso otros relojes con características similares al TSC pero de otras arquitecturas.

Comprobación de costo-beneficio entre la solución plasmada en este trabajo y la solución parcial física provista por el microprocesador: El análisis económico es importante para cualquier empresa que desarrolla microprocesadores para diferentes ambientes de aplicación. Actualmente en el mercado del rendimiento y el mercado del bajo consumo existe gran competencia entre fabricantes. Y esto ultimo afecta la selección o no de la solución debido a que la solución integra el ahorro de energía y la solución parcial física no y eso es una gran diferencia con respecto al mercado objetivo.

Adecuar la solución para que funcione en ambientes virtualizados y pueda ser utilizado como fuente de reloj: Actualmente en diferentes aplicaciones de virtualización no se puede seleccionar al TSC como origen de reloj, por más que el TSC disponga de la característica constant_tsc. La idea es proveer el uso del TSC en tales ambientes.

Sincronización a nivel de red local: Modificar la sincronización externa del nuevo reloj, para que realice una sincronización con un reloj externo de la PC. En este caso, se analizará el efecto de los ajustes sobre el nuevo reloj y la influencia del esquema de la red. Analizando la diferencia de la sincronización externa con un reloj general de la red o con un promedio de relojes enlazados directamente a la PC, midiendo convergencia horaria, error y escalabilidad.

Referencias

- Understanding The Linux Kernel, Daniel P. Bovet, Marco Cesati, 3rd Edition, O'Reilly November 2005, ISBN: 0-596-00565-2.
- Linux 2.6.35, arquitectura x86, http://www.kernel.org, código fuente del núcleo.
[1] Sistemas Operativos Distribuidos, A. S. Tanenbaum, 1ra Edición, Prentice Hall Hispanoamérica S.A., 1996.
[2] http://es.wikipedia.org/wiki/Condensador_de_alta_capacidad
[3] http://es.wikipedia.org/wiki/Oscilador_de_cristal
[4] http://es.wikipedia.org/wiki/Southbridge
[5] http://es.wikipedia.org/wiki/Ioctl
[6] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Intel, March 2010, Order Number: 253665-034US, capítulo 2.
[7] PCK2023 CK408 (66/100/133/200 MHz) Spread Spectrum Differential System Clock Generator, Philips Semiconductors, julio 2003.
[8] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Intel, March 2010, Order Number: 253668-034US, capítulo 8, subpagina 38.
[9] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, March 2010, Order Number: 253668-034US, capítulo 8, subpagina 35.
[10] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, March 2010, Order Number: 253668-034US, capítulo 8, subpagina 47.
[11] Front-side bus, http://es.wikipedia.org/wiki/Front-side_bus, Esta página fue modificada por última vez el 20 jul 2010.
[12] Everything You Need to Know About The QuickPath Interconnect (QPI), http://www.hardwaresecrets.com/article/Everything-You-Need-to-Know-About-The-QuickPath-Interconnect-QPI/610, By Gabriel Torres on August 25, 2008.
[13] Intel QuickPath Interconnect, http://es.wikipedia.org/wiki/Intel_QuickPath_Interconnect, Esta página fue modificada por última vez el 15 jun 2010.
[14] Software Optimization Guide for AMD64 Processors, AMD, September 2005, apendice A.
[15] BIOS and Kernel Developer's Guide for AMD AthlonTM 64 and AMD OpteronTM Processors, AMD, February 2006, capítulo 7.
[16] HyperTransport, http://es.wikipedia.org/wiki/HyperTransport, Esta página fue modificada por última vez el 16 feb 2010.
[17] Phase-Locked Loop Design Fundamentals, Garth Nash, Febrero 2006, Freescale Semiconductor.
[18] Phase-locked loop, http://en.wikipedia.org/wiki/Phase-locked_loop, This page was last modified on 23 July 2010.
[19] Phase Locked-Loop (PLL): Fundamento y aplicaciones, R. Pindado, Universitat Politècnica de Catalunya Departament d'Enginyeria Electrònica.
[20] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Intel, March 2010, Order Number: 253667-034US, capítulo 4.
[21] AMD64 Technology AMD64 Architecture Programmer's Manual Volume 2: System Programming, AMD, June 2010, capítulo 13, pag 346.

[22] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Intel, March 2010, Order Number: 253666-034US, capítulo 3, pag 197.
[23] AMD64 Technology AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, AMD, November 2009, capítulo 4. pag 294.
[24] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Intel, March 2010, Order Number: 253665-034US, capítulo 9.
[25] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Intel, March 2010, Order Number: 253668-034US, capítulo 8.2.5.
[26] Software Optimization Guide for AMD Family 10h Processors, AMD, May 2009, apendice A.
[27] AMD64 Technology AMD64 Architecture Programmer's Manual Volume 1: Application Programming, AMD, November 2009, capítulo 3, pag 92.
[28] AMD64 Technology AMD64 Architecture Programmer's Manual Volume 2: System Programming, AMD, June 2010, capítulo 7.6.3, pag 182.
[29] IA-PC HPET (High Precision Event Timers) Specification, Intel, October 2004.
[30] AMD SB700/710/750 BIOS Developer's Guide Technical Reference Manual, Advanced Micro Devices, Inc.
[31] Copmuter Network Time Syncronization, David L. Mills, 2da Edicion, 2011.
[32] http://en.wikipedia.org/wiki/Marzullo's_algorithm
[33] http://es.wikipedia.org/wiki/UTC
[34] http://es.wikipedia.org/wiki/Segundo_intercalar
[35] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Intel, March 2010, Order Number: 253668-034US, capítulo 14.
[36] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Intel, March 2010, Order Number: 253668-034US, capítulo 16, subpagina 48.
[37] Intel® 64 and IA-32 Architectures Optimization Reference Manual, Intel, November 2009, Order Number: 248966-020, capítulo 11.
[38] http://www.hardwaresecrets.com/article/611
[39] TSC and Power Management Events on AMD Processors, Brunner, Richard, http://lkml.org/lkml/2005/11/4/173, Ultima Modificacion 4 Nov 2005.
[40] Puente norte, http://es.wikipedia.org/wiki/Puente_norte, Esta página fue modificada por última vez el 15 jun 2010.
[41] AMD64 Technology Lightweight Profiling Specification, AMD, April 2010, capítulo2, pagina 21.
[42] Linux System Programming, Robert Love, 1rd Edition, O'Reilly Media, Inc, Septiembre 2007, capítulo 10.
[43] http://www.kernel.org/doc/man-pages/
[44] Linuxsymposium_procv1.pdf, Canada, Ontario, Julio 2005, pagina 219: A New Approach to timer and Timers.
[45] http://www.kernel.org/doc/ols/2006/ols2006v1-pages-333-346.pdf , 2006, Hrtimers and beyond: Transforming the Linux Time Subsystems.
[46] Documentacion Linux, highres.txt

[47] http://lwn.net/Articles/155862/, Octubre 2005, [ANNOUNCE] ktimers high resolution patches - clockevent abstraction layer
[48] http://lwn.net/Articles/223185/, Febrero 2007, Clockevents and dyntick
[49] http://www.networkworld.com/news/2007/022807-kernel.html, Febrero 2007, Clockevents y dyntick
[50] Linux Idle Power checkup, Len Brown, Aug 10, 2010, Intel Open Source Technology Center.
[51] ols2007v2-pages-119-126.pdf, Intel, 2007, cpuidle-Do nothing, efficiently...
[52] http://lwn.net/Articles/384146, Jonathan Coirbet, April 26, 2010, The cpuidle subsystem.
[53] http://www.lesswatts.org/projects/index.php, Salvando potencia con Linux sobre Plataformas Intel.
[54] Documentación Linux-cpuidle.
[55] Documentación Linux:cpufreq.
[56]http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq.html, Linux kernel cpufreq subsystem.
[57] http://how-to.wikia.com/wiki/How_to_configure_the_Linux_kernel/arch/i386/kernel/cpu/cpufreq, How to configure the Linux kernel/arch/i386/kernel/cpu/cpufreq.