Multiprocesamiento

Cristian Cuesta, Natalia Pérez, María Soldati, Alexis Urli, María Villa Duarte,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Universidad Nacional de la Matanza

Resumen. El multiprocesamiento hace referencia al uso de múltiples procesos concurrentes en un sistema, a diferencia del monoprocesamiento que ejecuta un único proceso en un instante determinado. Para llevar a cabo el multiprocesamiento, se debe contar con un planificador de procesos el cual se encarga de enviar los procesos a unos de los procesadores habilitados del sistema. Para lograr dicho alcance, se deben detectar la cantidad de procesadores del sistema operativo y conseguir su inicialización para que puedan ejecutar procesos concurrentes.

Palabras claves: Multiprocesamiento, planificador de procesos.

1. Introducción

El multiprocesamiento hace referencia al uso de múltiples procesos concurrentes en un sistema, a diferencia del monoprocesamiento que ejecuta un único proceso en un instante determinado. Para llevar a cabo el multiprocesamiento, se debe contar con un planificador de procesos el cual se encarga de enviar los procesos a unos de los procesadores habilitados del sistema.

Los sistemas multiprocesadores incorporan un fuerte acoplamiento al compartir la memoria y la capacidad de realizar interrupciones de entrada y salida. Es un sistema totalmente simétrico, es decir, que todos los procesadores son funcionalmente idénticos y de igual rango, y que cada uno de ellos puede comunicarse con todos los demás. No existe jerarquía, ni relación maestro-esclavo. Se destacan dos aspectos importantes:

• Simetría de memoria: La memoria es simétrica cuando todos los procesadores comparten el mismo espacio de memoria y el acceso a ese espacio por las mismas direcciones, es decir que posee la capacidad de que todos los procesadores puedan ejecutar una sola copia del sistema operativo. Cualquier proceso actual del sistema de software de aplicación se ejecutará en el mismo, independientemente del número de procesadores instalados en un sistema.

• E / S de la simetría: E / S es simétrica cuando todos los procesadores comparten el acceso al mismo subsistema I / O (incluidos los puertos de E / S y los controladores de interrupción) y un procesador puede recibir interrupciones desde cualquier fuente.

2. Líneas de investigación y desarrollo.

2.1 Características de los sistemas simétricos de multiprocesamiento (SMP) [ASD311]

Para mantener la coherencia de la memoria, debe existir algún protocolo de comunicación (o de acceso a memoria), y a veces es necesario permitir que un microprocesador bloquee (temporalmente) una dirección de memoria.

Para mantener la coherencia de la caché, cuando un microprocesador accede a la de otro, no debe recibir datos incorrectos (desactualizados). Ante cualquier cambio, todos los otros procesadores deben recibirlo.

Para distribuir las interrupciones cuando hay varios procesadores trabajando, es útil tener un mecanismo de control centralizado que las reciba, y las distribuya entre los diferentes microprocesadores.

2.2 Operaciones Atómicas [ASD311]

Estas operaciones son extremadamente necesarias en los SMP y se utilizan generalmente con estructuras compartidas (semáforos, descriptores de segmentos, páginas, etc.), cuando dos o más microprocesadores traten de modificar el mismo campo o flag. El procesador usa tres mecanismos para poder usarlas:

- Garantizar operaciones atómicas (algunas operaciones, como las lecturas y/o escrituras de un byte en memoria garantizan la ejecución atómica)
- Bloquear el bus (con la señal LOCK#, y el prefijo LOCK en la instrucción)
- Tener protocolos de coherencia de caché.

2.4 Advanced Programmable Interrupt Controller (APIC):

Cumple principalmente dos funciones para el procesador:

- Recibe las interrupciones, de fuentes internas y externas (I/O APIC u otros controladores de interrupciones externos), y las envía al core para manejarlas.
- En los sistemas MP, envía y recibe IPIs a y desde otros procesadores lógicos conectados al bus del sistema. Los mensajes IPIs pueden ser utilizados para distribuir interrupciones entre los procesadores o para ejecutar funciones propias del sistema (booteo, distribución del trabajo, etc.).

2.5 Protocolo de inicialización de multi-procesadores.

El protocolo define dos tipos de procesadores: BSP (bootstrap processor) y el AP (application processor). El hardware del sistema selecciona dinámicamente uno de los procesadores conectados al bus del sistema como el BSP; el resto queda designado como APs.

2.5.1 Requerimientos y restricciones del protocolo MP

El protocolo MP se ejecuta sólo después del encendido o un Reset; si se completó, y hay un BSP elegido, los INITs siguientes no provocan una nueva ejecución; en cambio, cada procesador examina su flag de BSP, para saber si debería ejecutar el código de boot-strap del BIOS, o si debería esperar por un SIPI.

2.5.2 Protocolo MP: mensajes IPI

SIPI: Startup IPI – este mensaje no causa ningún cambio en el estado pero produce que el procesador destinatario comience a ejecutar en modo real desde una dirección que contiene a un vector que forma parte del IPI.

BIPI: Boot IPI – inicializa el mecanismo arbitrario de selección del BSP en el grupo de procesadores del sistema que también designa al resto como de aplicación. Cada procesador enviará un BIPI al resto luego del apagado o reset del sistema.

FIPI: Final Boot IPI – comienza la inicialización del procedimiento del BIOS para el BSP. Este mensaje es enviado a todos los procesadores del sistema pero sólo el BSP debe responderlo comenzando con la ejecución del código de inicialización de la BIOS.

2.5.3 Secuencia típica de inicialización del BSP [ASD311]

1- Inicializa la memoria.

2- Carga el microcódigo en el procesador.

3- Inicializa los MTRRs (memory type range registers).

4- Habilita las cachés.

5- Ejecuta la instrucción CPUID con 0h en el EAX, y luego lee el EBX, ECX y EDX para determinar si el BSP es "Genuine Intel".

6- Ejecuta CPUID con 1h en el EAX, y luego almacena los valores de EAX, ECX y EDX en un espacio de configuración del sistema en la RAM para utilizarlo después.

7- Carga el código de arranque para que el AP ejecute en una página de 4 Kbyte en el Megabyte más bajo de memoria.

8- Cambia a modo protegido y se asegura que el espacio de direcciones APIC está mapeado en el tipo de memoria UC (strong uncacheable).

9- Determina el APIC ID del BSP del registro local de APIC ID.

10- Convierte la dirección base de la página de 4K del AP en un vector de 8 bits. El vector define la dirección de la página en el espacio de direcciones reales.

11- Habilita la APIC local configurando el bit 8 del SVR.

12- Configura la entrada de manejo de error LVT estableciendo un vector de 8 bits para el manejador de errores del APIC.

13- Inicializa la variable de Lock Semaphore. Los APs usan este semáforo para determinar el orden en que ejecutan el código de inicialización de APs de la BIOS.

14- Setea el BSP para detectar la presencia de APs en el sistema (y el número de procesadores) (Setea el contador a 1, larga un timer para que el AP incremente el contador).

15- Hace un broadcast de INIT-SIPI-SIPI IPI a los APs para despertarlos e inicializarlos.

16- Espera la interrupción del timer.

17- Lee y evalúa el contador y establece la cantidad de procesadores.

18- Si es necesario, reconfigura el APIC y continúa con los diagnósticos del sistema.

2.5.4 Secuencia típica de inicialización del AP [ASD311]

Cuando un AP recibe una SIPI, realiza los siguientes pasos:

1- Espera al Lock Semaphore de inicialización de la BIOS. Una vez que consigue su control, continúa la inicialización.

2- Carga el microcódigo en el procesador.

3- Inicializa los MTTRRs (utiliza el mismo maping que el BSP).

4- Habilita la caché.

5- Ejecuta CPUID con 0h en el EAX, y luego lee los otros registros para comprobar que el AP sea "Genuine Intel".

6- Ejecuta CPUID con 1h en EAX, y guarda EAX, ECX y EDX en un espacio de configuración del sistema en la RAM para utilizarlo posteriormente.

7- Cambia al modo protegido y asegura que el espacio de direcciones del APIC está mapeado como uncacheable.

8- Determina el APIC ID propio del registro local de APIC ID, y lo agrega a la tabla de ACPI y MP.

9- Inicializa y configura el APIC local estableciendo el bit 8 del SVR y seteando el LVT3 para el manejo de errores.

10- Configura el ambiente de ejecución SMI. (Cada AP y el BSP deben tener una dirección SMBASE diferente).

11- Incrementa el contador en 1.

12- Libera el semáforo.

13- Ejecuta instrucciones CLI y HLT.

14- Espera por un INIT IPI.

2. 6 ICR – Interrupt Command Register

Las IPI's se envían mediante la escritura de los registros del ICR, lo cual funciona como un inicializador para los procesadores de aplicación. Los principales son:

ESR Error Status Register: el ESR almacena todos los errores detectados por el APIC local.

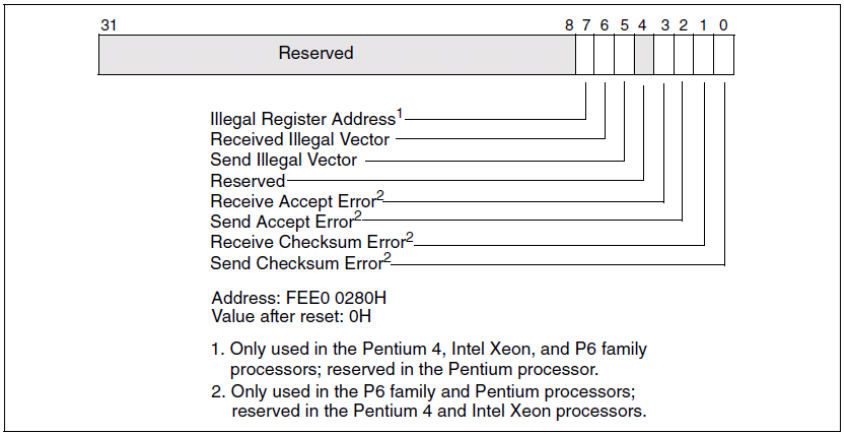

Imagen 1. Registro de Estado de Error (ESR)

LVT LINT0 Register (FEE 0350H): Especifica la entrega de la interrupción cuando se señala una en el pin LINT0.

LVT LINT1 Register (FEE0 0360H) : Especifica la entrega de la interrupción cuando se señala una en el pin LINT1.

LVT Error Register (FEE0 0370H) : Especifica la entrega de una interrupción cuando el APIC detecta un error interno.

TPR Task Priority Register: El APIC local también define una prioridad para cada tarea y para cada procesador que se utiliza para determinar el orden en que se deben manejar las interrupciones. La prioridad de la tarea es un valor del software seleccionado entre 0 y 15 que se escribe en la tarea prioritaria de registro TPR, el cual es un registro de lectura/escritura.

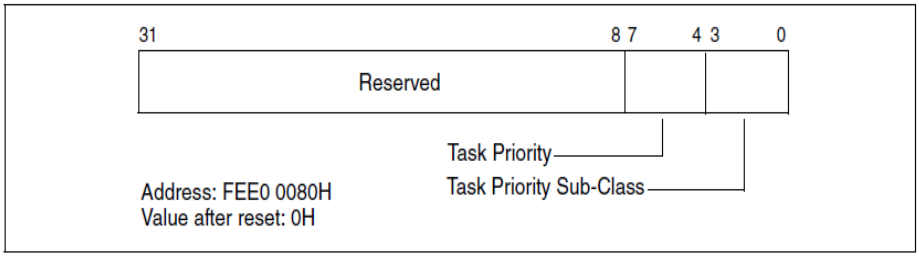

Imagen 2. Registro de Prioridad de Tareas (TPR)

SVR Spurius Interrupt Vector Register : se inicializa en 000000FFH. Mediante el establecimiento de 8 bits a 0 el software deshabilita el APIC local.

DCR Divide Configuration Register: La unidad de APIC local contiene un temporizador programable de 32 bits que está disponible para el software de tiempo, eventos o actividades. El mencionado registro es necesario para la programación de dicho temporizador.

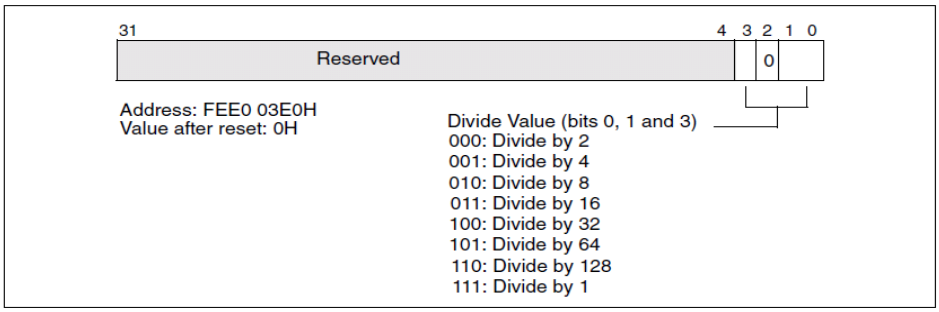

Imagen 3. Registro de Configuración de la división (DCR)

LDR Local Destination Register: utilizado en conjunción con el modo de destino lógico y la dirección de destino del mensaje para seleccionar los procesadores de destino.

DFR Destination Format Register: de igual función que el registro anterior, se utiliza en combinación con el modo de destino lógico y la dirección destino del mensaje para seleccionar los procesadores de destino.

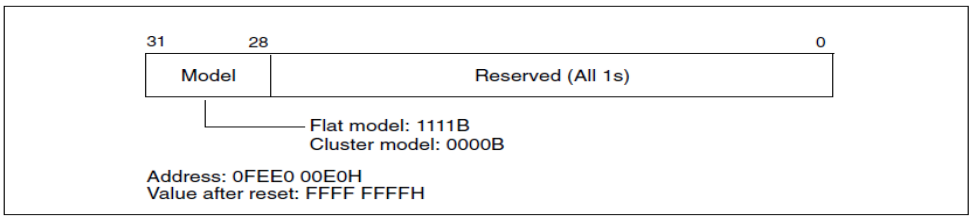

Imagen 4. Registro de Formato de Destino (DFR)

2.7 Detección de la Tabla MP y Configuración

Cuando el sistema se inicializa, el BIOS detecta el hardware instalado y crea estructuras para describirlo. Existen dos tablas relacionadas al sistema simétrico de multiprocesamiento. Las mismas son MP Floating Pointer Structure, requerida obligatoriamente, y la MP table, la cual es opcional. Si la última mencionada no existe, el sistema deberá establecer la configuración por defecto definida en la primera estructura.

En primer lugar, es necesario encontrar la estructura de puntero flotante. De acuerdo a la especificación la misma puede hallarse en uno de los siguientes cuatro lugares:

- El primer kb del área extendida del BIOS
- El último kilobyte de la memoria base
- La parte superior de la memoria física
- La memoria de sólo lectura de la BIOS que se encuentra comprendida entre 0xE0000 y 0xFFFFF.

Para confirmar su presencia, es necesario buscar en estas áreas los cuatro bytes de la firma "_MP_", que indica el principio de la estructura. Su ausencia determina que el sistema no es compatible con

multiprocesamiento, en cuyo caso el sistema operativo puede detener la ejecución o utilizar la configuración establecida para un único procesador.

2.8 Utilización del APIC local

Los sistemas de multiprocesamiento poseen un bus especial para que todos los APIC en el sistema estén conectados y puedan comunicarse entre sí dado que la memoria es compartida. Los APIC son dispositivos de memoria mapeada. La ubicación predeterminada para el local se encuentra en 0xFEE00000 de la memoria física. El software puede acceder a los APIC ID de las siguientes formas:

- Leer el APIC ID en el APIC local.
- Leer la tabla ACPI o la MP.
- Leer el APIC ID inicial.
- Leer el APIC ID con CPUID leaf 0Bh.

2.9 Inicialización de los Procesadores de Aplicación

Un AP podría ser inicializado ya sea por el BSP o por otro AP activo. Si la tabla de configuración MP existe, ésta provee los IDs de los APICs locales pertenecientes a los AP. Estos IDs serán utilizados posteriormente como direcciones de destino en el envío de interrupciones.

Una vez que se han detectado los procesadores en el sistema, seteado el local APIC y verificado que se ha establecido comunicación con el mismo, es el momento de bootear los APs. Como los mismos se despertarán en modo real, todo lo que necesitarán para comenzar debería estar en memoria baja (debajo de 0x100000 o 1MB).

El procedimiento consiste en enviar una secuencia de interrupciones al procesador. En primer lugar, se debe enviar una INIT IPI, afirmando la señal mediante la escritura del APIC ID del procesador de destino en la parte alta del ICR y escribiendo en la parte baja los bits para habilitar el modo de entrega en INIT, nivel de activación y confirmación de la interrupción. El APIC local indica que el envío de una IPI fue exitoso reseteando el bit de estado de entrega (delivery status) en el ICR.

Si se tuvo éxito con lo explicado arriba, el procesador en algún momento despertó en modo real y comenzó a ejecutar el código que le fue indicado. En primer lugar, se debe ejecutar una instrucción "cli" para deshabilitar las interrupciones, para luego pasar a modo protegido.

2.10 GDT (Global Descriptor Table)

Cuando se opera en modo protegido, todos los accesos a memoria pasan a través de la tabla global de descriptores (GDT) o de la tabla local de descriptores (LDT). Estas tablas contienen registros llamados descriptores de segmentos. Un descriptor de segmento provee la dirección base del segmento, así como los permisos de acceso, tipo e información de uso. Cada descriptor de segmento tiene asociado un selector de segmento. Este provee el software que lo usa con un índice en la GDT o LDT (el offset de su descriptor de segmento asociado), un flag que indica "global" o "local" e información de permisos de acceso.

Para acceder a un byte en un segmento, se deben suministrar un selector de segmento y un offset.

El selector de segmento provee acceso al descriptor para el segmento (en la GDT o LDT). El procesador obtiene del descriptor de segmento la dirección base del segmento en el espacio de direcciones lineales. Este mecanismo puede ser usado para acceder a cualquier código válido, datos o segmento de pila, siempre que el segmento sea accesible desde el nivel de privilegio actual (CPL) en el que el procesador está funcionando. El CPL se define como el nivel de protección del segmento de código que se está ejecutando actualmente. El nivel para el sistema operativo es el anillo 0 y para el usuario, el 3. La mayoría de los sistemas operativos utilizan los anillos 0 y 3.

En la figura que sigue, las flechas continuas indican una dirección lineal, las líneas de puntos un selector de segmento, y las flechas de puntos indican una dirección física. Para simplificar, muchos de los selectores de segmento se muestran como indicadores directos de un segmento.

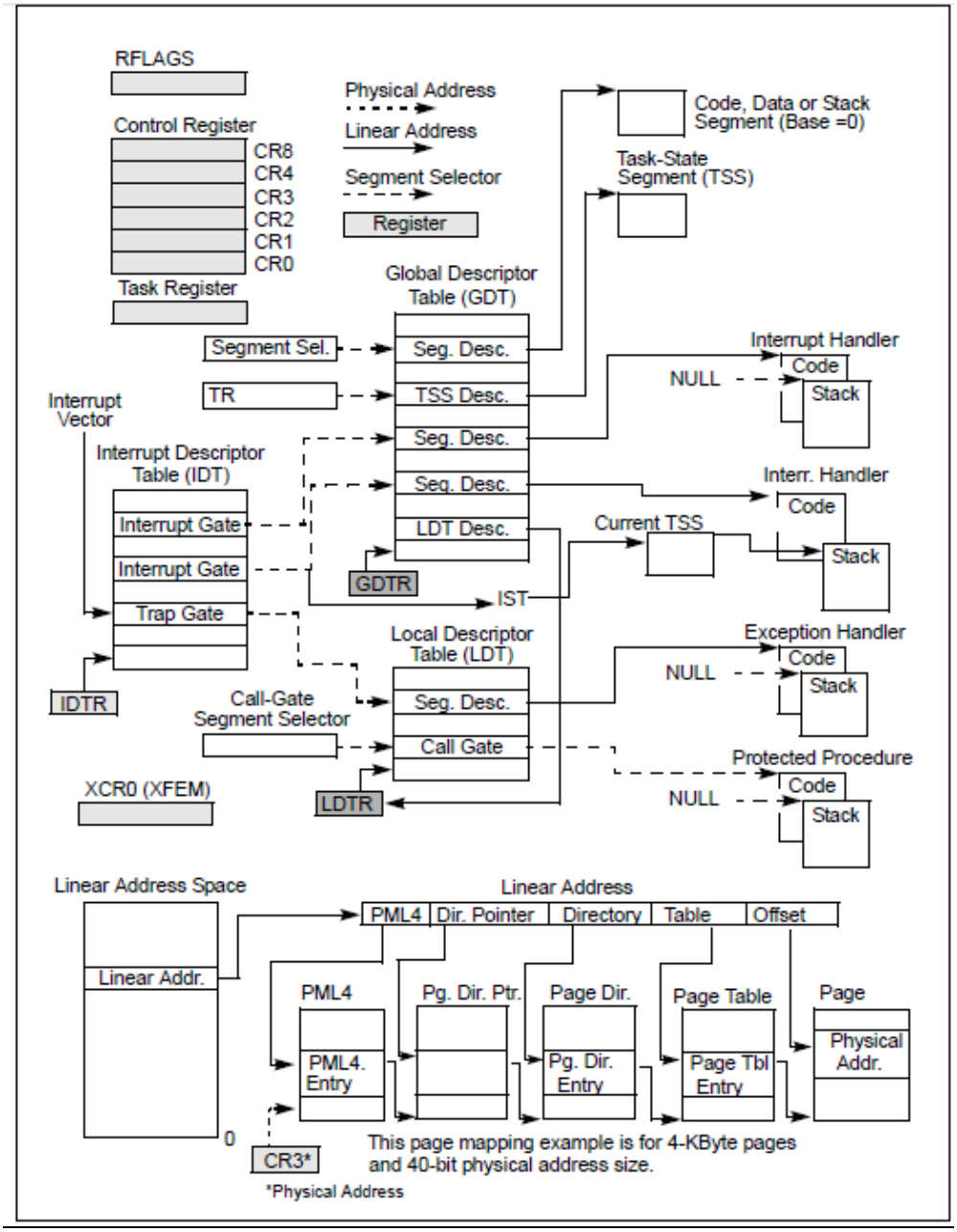

Imagen 5. Registros de nivel de Sistema y Estructuras de datos.

3. Resultados obtenidos/esperados.

Se destaca que la designación del BSP y la inicialización de todos los procesadores son realizadas automáticamente. En este punto el BSP se encuentra inicializando el sistema operativo y los procesadores de aplicación en estado de espera aguardando por una señal. El primer problema a resolver es obtener el APIC ID de cada procesador físico del sistema. Como se mencionó anteriormente, existen diferentes formas de obtenerlo; la elegida para este desarrollo fue leer la tabla MP [IMS97]. Se busca el descriptor que es una

estructura de tamaño variable en múltiplos de 16 bytes. Para reconocerlo, el primer campo del descriptor es la firma "_MP_". Una vez que se encuentra la firma, se continúa leyendo los siguientes 32 bits para conseguir la dirección de la tabla MP. En el caso de que esta dirección estuviera formada por ceros únicamente, significa que la tabla MP no existe, y por lo tanto los APIC ID de los diferentes procesadores estarán obligatoriamente asignados de manera consecutiva.

Una vez conseguida la dirección de la tabla, comienza su lectura. La misma cuenta también con una firma ("PCMP"), que avala su presencia. Los siguientes 16 bits indican el tamaño de la misma (incluyendo la firma); luego, los próximos 8 bits indican la versión de la especificación de la MP (puede ser versión 1.1 o versión 1.4), entre otros. Entre ellos se encuentra la dirección del APIC local: es donde se indica que cada procesador accede a sus APIC local.

Una vez que la tabla termina, le sigue un número variable de entradas. El primer byte de cada una identifica el tipo. Cada entrada cuenta con la siguiente información: Procesador (única por procesador), Bus, APIC de E/S, Asignación de interrupción de E/S, y Asignación de interrupción local.

La entrada de los procesadores indica de cuál se trata (0 si es procesador), el APIC ID local, si el procesador es o no accesible, la firma del procesador (stepping, modelo y familia).

Se definieron en smp.h, entre otras macros necesarias para establecer la disponibilidad de los procesadores, fundamentalmente las siguientes variables y funciones:

#define SMP_MAGIC_IDENT ('_'<<24)|('P'<<16)|('M'<<8)|'_'), que es la firma a encontrar para confirmar la presencia de la tabla de multiprocesamiento.

#define MPC_SIGNATURE "PCMP", que es una firma errónea que puede hallarse en confusión con la anterior

Las entradas de la Tabla MP, correspondientes a los tipos de entrada que pueden hallarse en la lectura: #define MP_PROCESSOR 0, #define MP_BUS 1, #define MP_IOAPIC 2, #define MP_INTSRC 3, #define MP_LINTSRC 4

int iSoporteMP; necesaria para continuar con la ejecución normal en caso de no hallar más de un procesador

int iCantProcesadores; cantidad de procesadores con la que cuenta el sistema

int iDisponibilidadProcesadores[MAX_PROCESADORES]; procesadores disponibles para trabajar (indicando OCUPADO o LIBRE según sea el caso)

int iApicId[MAX_PROCESADORES]; APIC ID de cada procesador obtenido de la lectura de la tabla

int iMasterId; determina al BSP.

Las siguientes funciones se encuentran desarrolladas en smp.c:

int iFnSmpScanConfig(unsigned long , unsigned long); busca el descriptor en las diferentes áreas de memoria detalladas anteriormente

static int siFnMPChecksum(unsigned char *, int); una vez que el descriptor es hallado, procede a verificar que la firma hallada es la correcta

static int siReadMPTable(struct stuMPConfigTable *stuMpc); lee cada entrada de la tabla MP, almacenando el APIC ID de los procesadores y estableciendo la cantidad disponible.

void vFnConfigurarSMP(); de acuerdo a lo hallado en la función anterior asocia la APIC ID con su estado para la ejecución.

La lectura de la tabla MP proporcionó el conocimiento de la cantidad de procesadores físicos disponibles para trabajar, pero el problema se centra en que los mismos se encuentran esperando su activación. Según la información provista por el manual de Intel, los APs han quedado en este momento en un estado de halt, por lo que es necesario enviarles una IPI para indicarles que pueden comenzar a ejecutar. Una vez logrado esto, se les debe enviar una tarea para que continúen su ejecución.

El primer inconveniente se presentó al tratar de enviar las IPIs. Se siguió y buscó el código de la función smp_boot_cpus () del kernel de Linux [LXR06]; este intento falló porque es necesario que el kernel acceda a una posición de memoria alta para poder ubicar en qué dirección se encuentra el APIC local. En Linux esto se hace a través de la función vremap(), la cual se maneja con paginación que se encuentra implementada de manera diferente a la del SODIUM. Al tratar de implementar una función propia que supliera a vremap(), se presentó el inconveniente que la dirección de memoria que busca no está definida, por lo tanto no se puede acceder ya que no se sabe cuál es con exactitud.

Seguidamente, se investigó el sistema operativo Helen OS [HOS06], el cual presentaba una serie de funciones para la inicialización de los procesadores de aplicación que fueron adaptadas e incorporadas en SODIUM.

Las funciones encargadas de la inicialización quedaron determinadas en los archivos smp.h y smp.c, y son las siguientes:

vFnIniciarSMP: Encargada de llamar a las funciones que envían las señales IPI.

vFnEnviarSeniales: crea la GDT para cada procesador (se determinó que para cada uno es necesaria una gdt con los procesos que pueda ejecutar, para imitar el comportamiento de la función de Linux set_affinity()), y llama a la función que inicializa el APIC local. Una vez que esta termina, vFnEnviarSeniales llama a iFnEnviarInitIPI, que indica el modo de envío (escribiendo los registros del ICR), y la señal IPI es generada y depositada en el bus del sistema.

vFnInicializarApicLocal: esta función se encarga de escribir en los registros del ICR, inicializándolos.

La investigación continuó por la línea de la modificación de la estructura de la GDT, para poder adaptarla al multiprocesamiento; el desarrollo consistió en crear un vector de estructuras, según la cantidad máxima de procesadores que el sistema pueda manejar, y la misma se inicializó según la cantidad de procesadores detectados en el momento de lectura de la tabla MP. Se dejó asociada la primer posición del vector al BSP, y se instanció las otras posiciones, aunque por ahora no es posible asignarles tareas.

En SODIUM, tenemos el archivo gdt.c, que contiene las rutinas de administración de la GDT y de creación de procesos. A su vez, mantiene el último PID utilizado, la tabla de PCBs y la TSS.

Para la implementación de la GDT con varios procesadores, lo que se hizo fue, en primera instancia, crear un vector con un máximo de ocho posiciones, en cada una de las cuales tiene una estructura gdt. Cada una de ellas corresponde a cada procesador (incluyendo el BSP y todos los AP). Entre los datos que integran la estructura gdt, se encuentran la dirección base del segmento, la longitud del mismo, permisos de acceso y demás elementos que identifican a cada segmento.

4. Conclusiones

A pesar de que la inicialización de los procesadores de aplicación está realizada, la ejecución de tareas en sistemas multiprocesamiento no ha podido ser terminada, principalmente por la falta de ciertos recursos esenciales. Idealmente, el proceso de inicialización debería realizarse con un hilo de kernel que recorra un vector que posee su propia pila dedicada. Esto se produce porque la pila utilizada por el BSP durante su inicialización será usada luego para la inicialización de las correspondientes a cada AP. Es necesario que cada uno de ellos se despierte a través de un hilo, que será quién controle que los mismos se despierten, que comiencen a ejecutar correctamente y los errores en caso de haberlos.

Como se explicó anteriormente, para poder migrar a SMP, es necesario garantizar la ejecución de operaciones atómicas. SODIUM no cuenta con la implementación de ningún tipo de estructura que las asegure, por lo cual es requerido que antes del cambio puedan implementarse semáforos. Esto se produce porque debe haber un único AP inicializándose en un período de tiempo. Luego de su cambio de estado, debe continuar el resto.

Se debe configurar y modificar por completo el sistema de interrupciones debido a que cada procesador trabajará independientemente y puede detener o reanudar la ejecución en distinto período de tiempo que el resto. Además, al tener múltiples procesadores trabajando al mismo tiempo, es necesario identificar a quién van dirigidas las interrupciones que se reciben, por lo cual es necesario administrarlas en forma diferente a la de los sistemas monoprocesador.

4. Bibliografía

[ASD311] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A & 3B): System Programming Guide, Mayo 2011.

[GRB09] Guía GRUB http://www.guia-ubuntu.org/index.php?title=GRUB

[HOS06] HelenOS http://www.helenos.org

[IBM10] IBM developerWorks http://www.ibm.com/developerworks/linux/library/l-affinity/index.html

[IMS97] Intel® MultiProcessor Specification Version 1.4, 1997.

[IPI11] Intel® Processor Identification and the CPUID Instruction, January, 2011.

[LXR06] The LXR Cross Referencer (Linux/arch/alpha/kernel/smp.c).

[IRP93] Intel® Reveals Pentium Implementation Details, 1993.