DeliriOS: Exokernel Multicore en 64bits

Silvio Vilariño, Ezequiel Gambacini

15 de julio de 2014

Delirios: Exokernel 64 bits Multicore

Motivación del trabajo final

- La **idea** directora del proyecto era experimentar con la arquitectura Intel
- Para esto se desarollo un exokernel de 64 bits **multinúcleo**
- El objetivo inicial fue con **fines didácticos**, posteriormente se implementaron experimentos sobre la plataforma

Delirios: Exokernel 64 bits Multicore

Características

Se desarrollo un exokernel en 64 bits,

- El sistema permite ejecutar una única tarea en varios procesadores
- El *overhead* del OS es mímimo
- No hay soporte de protección, todo corre en nivel 0
- No hay soporte de entrada-salida

Desarrollo
It's working!

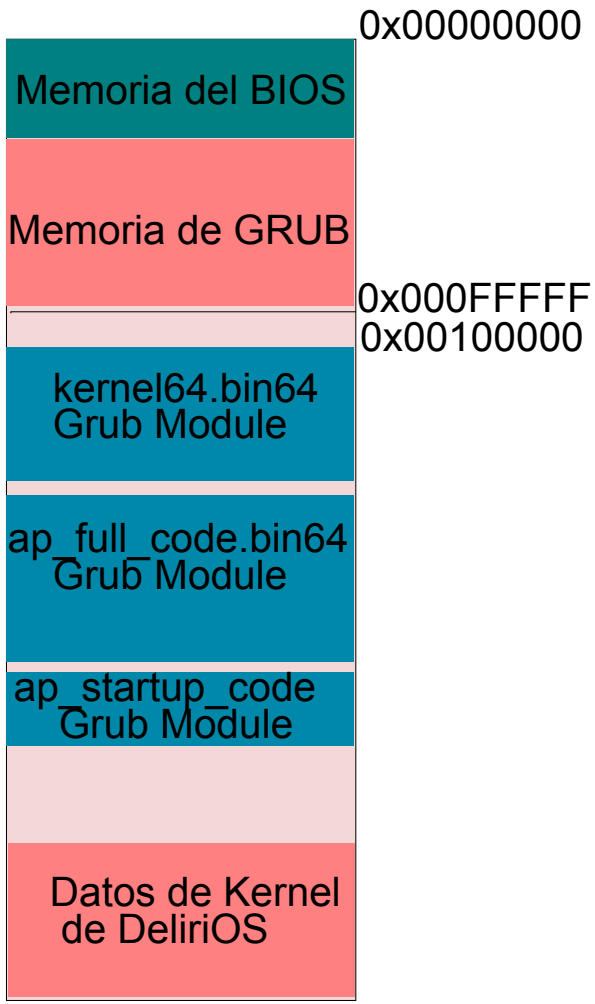
- El desarrolo consistio en estudiar todo el proceso de inicio en 64bits e implementarlo
- Además se utilizo `grub` como bootloader para poder iniciar desde un pendrive y sobre una máquina física
- Por ultimo se implemento sobre dos procesadores un algoritmo para ordenar elementos

Desarrollo
Inicialización de `grub`

- La revisión de `grub` no permite ejecutables en 64bits
- Por lo que se utilizo la carga de módulos
- Consiste en cargar partes del sistema sobre el primer megabyte
- `Grub` permite determinar las posiciones de los módulos

Desarrollo

Mapa de memoria: Memoria baja y módulos en memoria alta



Desarrollo

Niveles de booteo del BSP y preparación del entorno de inicio de los AP

Grub inicializa la máquina a un estado conocido y otorga el control al loader de nivel 1 del BSP pasándole por parámetros estructuras de grub con información del sistema.

1. Se realizan **validaciones** requeridas por la especificación multiboot, verificación de firmas, etc.
2. Se obtienen de la metadata provista por `grub` las posiciones de memoria **donde están cargados** los módulos, ellos son:

kernel64.bin64:
: segundo nivel de **booteo del BSP** (binario plano de 64 bits)

ap_full_code.bin64:
: segundo nivel de **booteo de los Application Processors** (binario plano de 64 bits)

ap_startup_code:
: código de inicio del **AP en modo real y el primer stage** de booteo a modo protegido (binario de 32 bits)

Desarrollo

Niveles de booteo del BSP y preparación del entorno de inicio de los AP (cont...)

1. Los AP inician en modo real $\rightarrow$ deben comenzar su ejecución por debajo del primer mega de memoria principal $\rightarrow$ se copia el módulo `ap_startup_code` a una dirección arbitraria alineada a página `0x2000`. (se superpone código con `grub`)

Desarrollo

Niveles de booteo del BSP y preparación del entorno de inicio de los AP (cont...)

1. Los AP inician en modo real $\rightarrow$ deben comenzar su ejecución por debajo del primer mega de memoria principal $\rightarrow$ se copia el módulo `ap_startup_code` a una dirección arbitraria alineada a página `0x2000`. (se superpone código con `grub`)

2. Para evitarlo, minimizamos el tamaño del startup de modo real del AP, haciendo lo antes posible un salto a un loader de nivel 2 por encima del mega (`ap_full_code.bin64`)

Desarrollo

Niveles de booteo del BSP y preparación del entorno de inicio de los AP (cont...)

1. Los AP inician en modo real $\rightarrow$ deben comenzar su ejecución por debajo del primer mega de memoria principal $\rightarrow$ se copia el módulo `ap_startup_code` a una dirección arbitraria alineada a página `0x2000`. (se superpone código con `grub`)

2. Para evitarlo, minimizamos el tamaño del startup de modo real del AP, haciendo lo antes posible un salto a un loader de nivel 2 por encima del mega (`ap_full_code.bin64`)

3. El AP debe saltar entre los dos niveles de booteo $\rightarrow$ se le inyecta al primer módulo la dirección donde esta cargado el segundo nivel de booteo

Desarrollo

Niveles de booteo del BSP y preparación del entorno de inicio de los AP (cont...)

1. Los AP inician en modo real $\rightarrow$ deben comenzar su ejecución por debajo del primer mega de memoria principal $\rightarrow$ se copia el módulo `ap_startup_code` a una dirección arbitraria alineada a página `0x2000`. (se superpone código con `grub`)

2. Para evitarlo, minimizamos el tamaño del startup de modo real del AP, haciendo lo antes posible un salto a un loader de nivel 2 por encima del mega (`ap_full_code.bin64`)

3. El AP debe saltar entre los dos niveles de booteo $\rightarrow$ se le inyecta al primer módulo la dirección donde esta cargado el segundo nivel de booteo

4. Finalmente, se realiza un salto en la ejecución a donde comienza el modulo `kernel64.bin64` donde el BSP finaliza la inicializacion del contexto hasta modo largo de 64 bits y enciende a los demás núcleos del sistema

Desarrollo
Inicializacion por niveles BSP

Grub lleva pc a un estado conocido detallado en la especificación multiboot

nivel 1: Puente entre Grub y DeliriOS

* Validaciones de especificacion multiboot.
* Obtención de las posiciones en memoria de los módulos.
* Copia del módulo ap_startup_code a la posición 0x2000.
* Inyección de datos entre módulos.
* Otorgar control al módulo kernel64

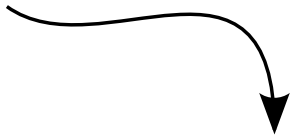
nivel 2: Puente entre DeliriOS y modo x64

* Asignar GDT, Paginación, Interrupciones, etc.
* Reprogramación del PIC.
* Envío de señal de encendido a los AP's .
* Limpieza de registros de propósito general.
* Seteo de la pila.
* Salto a 64 bits

Inicio completo!

Desarrollo
Inicializacion por niveles AP

Ap bootea en 0x2000 en modo real
* Creación de GDT temporal en código
* Salto de modo real a modo protegido
* Salto a loader de nivel 2



nivel 2: Modo protegido a modo x64
* Asignar GDT, IDT, Paginación
* Seteo de la pila.
* Salto a modo x64.

Inicio completo!

Desarrollo: Inicialización del Bootstrap Processor

Modo Legacy x64: Modelo de segmentación

QUIEN ASEGURA ??? explicar mejor o detallar en pasos

La especificación *multiboot* nos asegura que estamos en modo protegido, pero no tenemos la certeza de tener asignada una GDT válida, es por esto que asignamos una GDT con 3 descriptores de nivel 0, una para datos y otras dos de código, esta diferenciación de descriptores de código es necesaria para realizar los jump far para pasar de modo real hacia modo protegido y de modo protegido-compatibilidad x64 hacia modo largo x64.

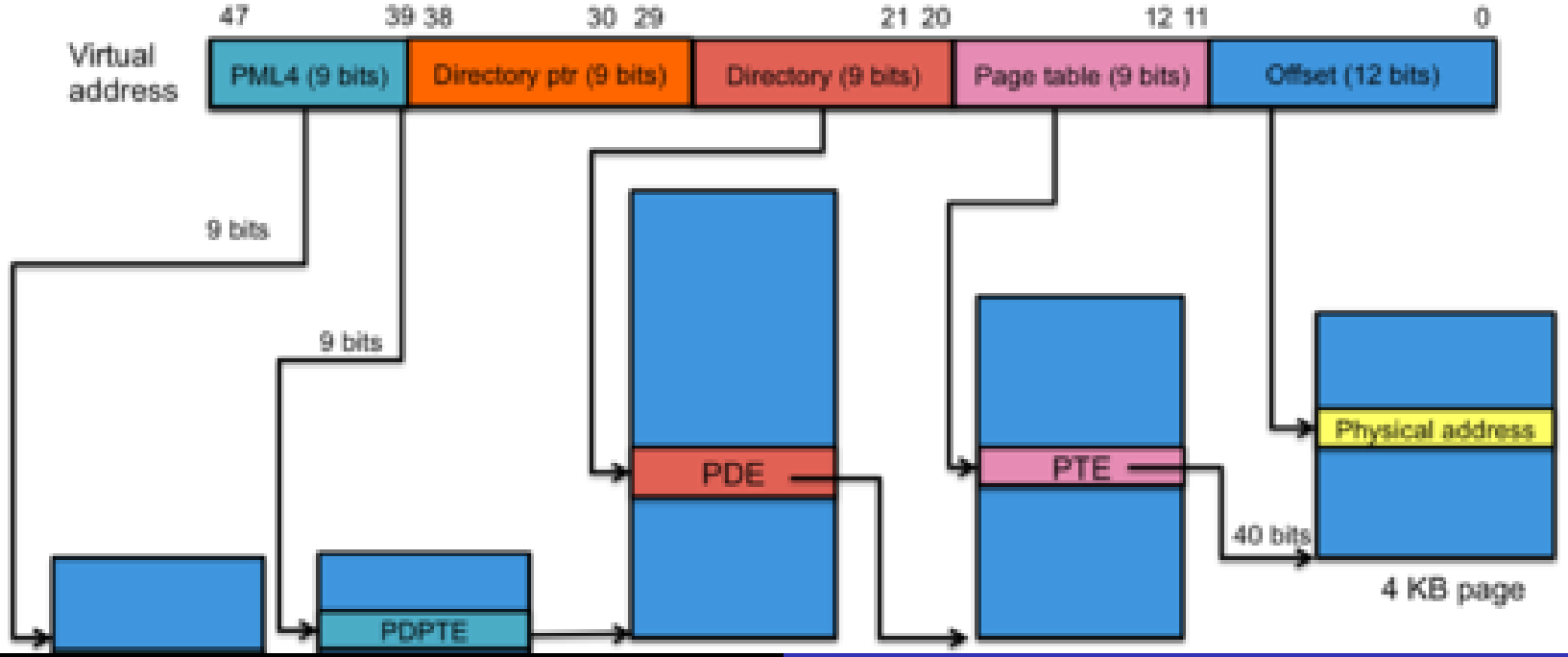
Índice	Descriptor
0	Descriptor nulo
1	Código nivel 0 para 32 bits
2	Código nivel 0 para 64 bits
3	Datos nivel 0 para 32 y 64 bits

Desarrollo: Paginación en 64 bits

x64 requiere habilitar PAE

Se realizo un mapeo identity mapping de los primeros 4gb usando paginas de 4kb.

este dibujo es un espanto!!! buscar otro que tenga mas de 4pixeles!



Desarrollo: Verificacion de disponibilidad de x64

Consulta via `cpuid`

Luego de establecer estas estructuras, realizamos una comprobación vía `cpuid` para saber si está disponible modo x64,

de verificarse esta comprobación, encendemos los bits del procesador correspondientes para habilitar dicho modo en el registro CR0.

Desarrollo: Pasaje a modo largo x64 nativo

Para pasar de modo compatibilidad a modo nativo de 64 bits, es necesario realizar un salto largo en la ejecución a un descriptor de la GDT de código de 64 bits.

Luego de realizar el salto al segmento de código de x64 de la GDT establecemos un contexto seguro con los registros en cero, seteamos los selectores correspondientes de la GDT y establecemos los punteros a una pila asignada al BSP.

Desarrollo: Inicialización del PIC

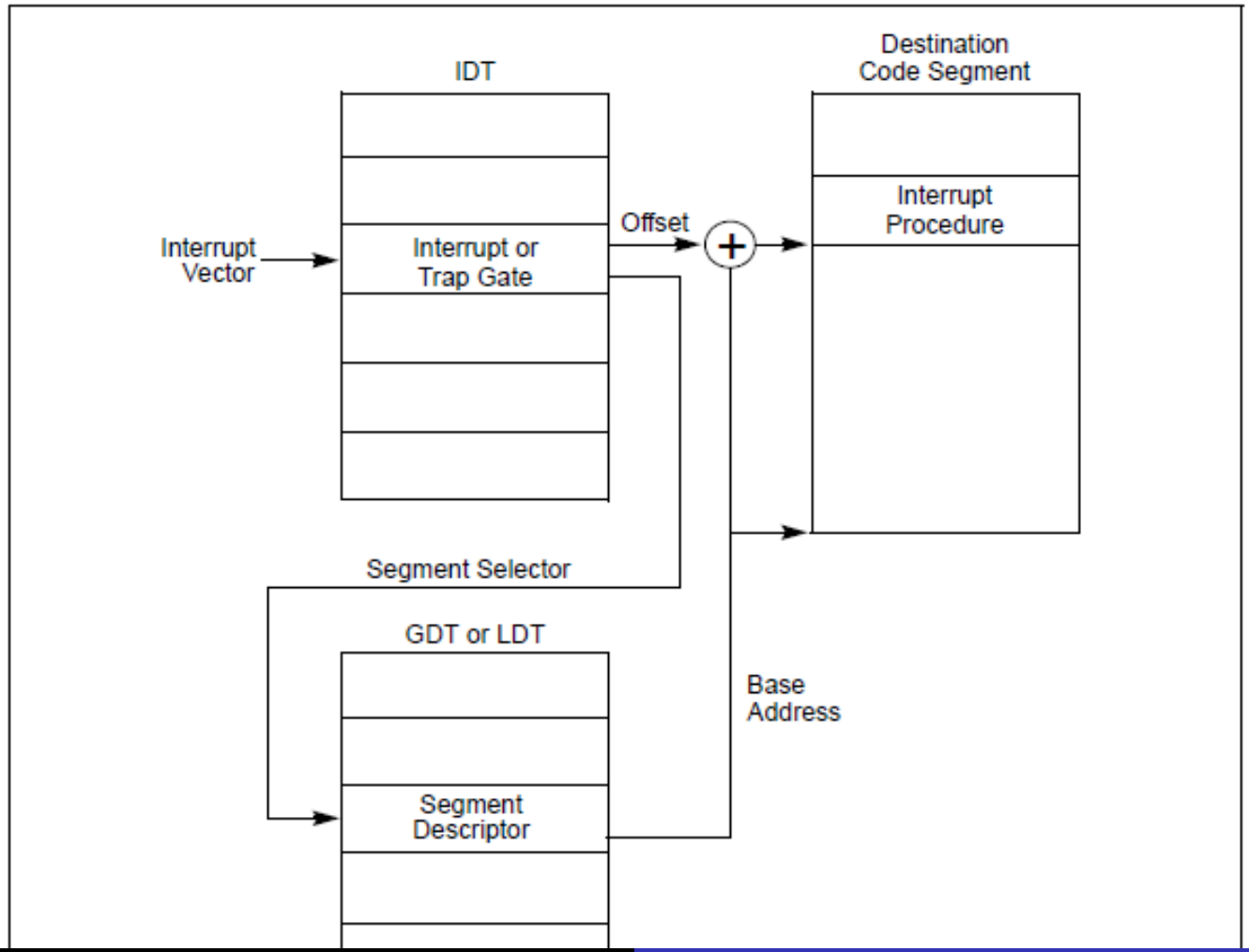
Captura de excepciones e interrupciones

Enviamos señales al PIC para reprogramarlo de forma tal en la que atienda las interrupciones enmascarables.

Luego, asignamos una IDT que captura todas las excepciones e interrupciones y de ser necesario, realiza las acciones correspondientes con su ISR asociada.

Particularmente las excepciones son capturadas y mostradas en pantalla y se utiliza la interrupción de reloj para sincronización y esperas, las demás interrupciones son ignoradas. esta frase no tiene sentido. o si?

que onda este dibujo? porque esta, que explicacion tiene?



Desarrollo: Multicore - encendido de los AP's

Una vez inicializado el bsp, se procede a inicializar el resto de los procesadores del sistema.

Para lograr esto, primero es necesario que encontrar una estructura de datos llamada `MP Floating Pointer Structure`, que contiene:

- información sobre los demás procesadores
- apic (advanced programable interrupt controller)
- el I/O apic (análogo al apic, pero se encarga además de rutear interrupciones de input/output a los lapic's)

Desarrollo: Búsqueda de la estructura `MP Floating Pointer`

Esta estructura puede estar en diferentes lugares de memoria, inicialmente se debe realizar la búsqueda dentro del **primer kilobyte de la ebda** (extended bios data area),
de no encontrarse allí, se procede a buscar entre los **639K-640K** de memoria.

- Para encontrar la tabla se busca dentro de esas áreas de memoria la firma de la `MP Floating Pointer Structure`, la cual es "`_MP_`"
- Para comprobar la validez de la estructura se realiza un checksum
- Si no se encuentra la tabla, el sistema no soporta multicore

Desarrollo: Inicialización de los AP's

Una vez finalizada la inicialización del `local apic`, se debe pedir a la BIOS que setee el `warm reset vector` a la dirección donde esta localizado el inicio de modo real de los aps($0x2000$).

Luego de este paso, se procede a encender los AP's usando la información obtenida por la `MP Configuration Table`.

Desarrollo: Inicialización de los AP's (cont)

Luego de tener listos los `icr`, se procede a mandar las ipi's de INIT e INIT_DASSERT, luego se espera unos 10 milisegundos, verificando previamente que las ipis se hayan enviado correctamente.

Luego de la espera, se procede a enviar la `ipi` de STARTUP, se espera 10 ms, se verifica que se haya enviado correctamente, se la vuelve a enviar, se realiza otra espera de 10 ms, y se vuelve a verificar.

Una vez terminado este proceso, se puede asumir que el AP fue encendido, y se continúa el encendido el resto de los AP's encontrados en la MP Configuration Table

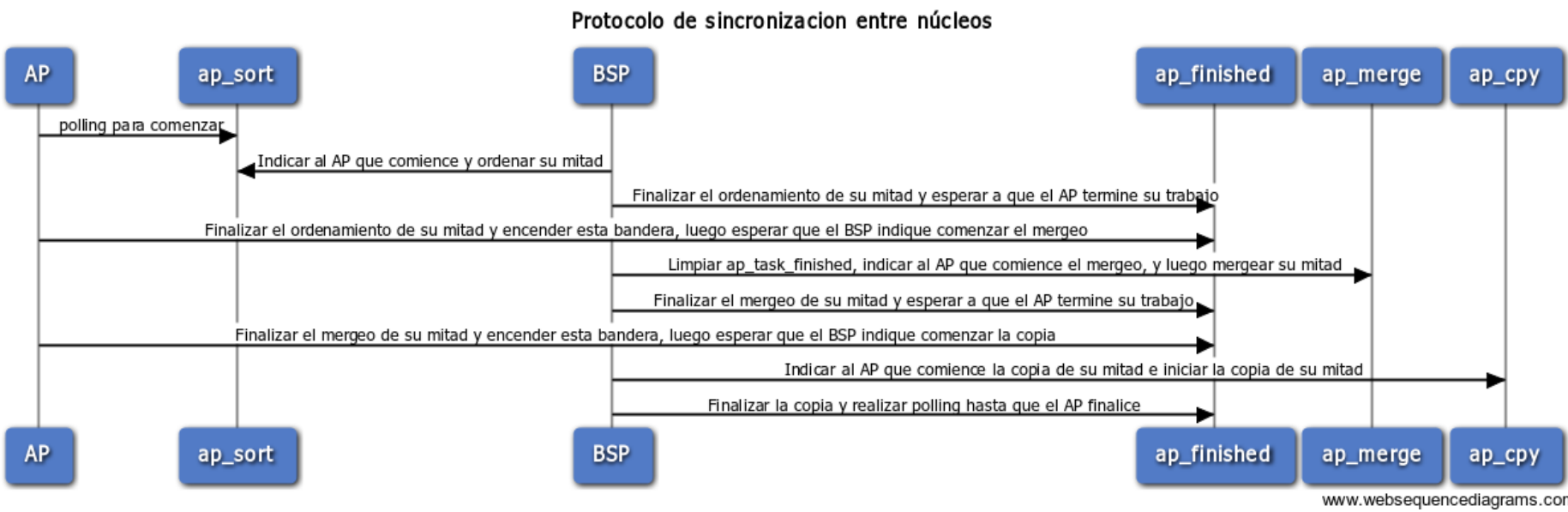
Multicore: inicialización de modo real a modo nativo x64 de los AP's

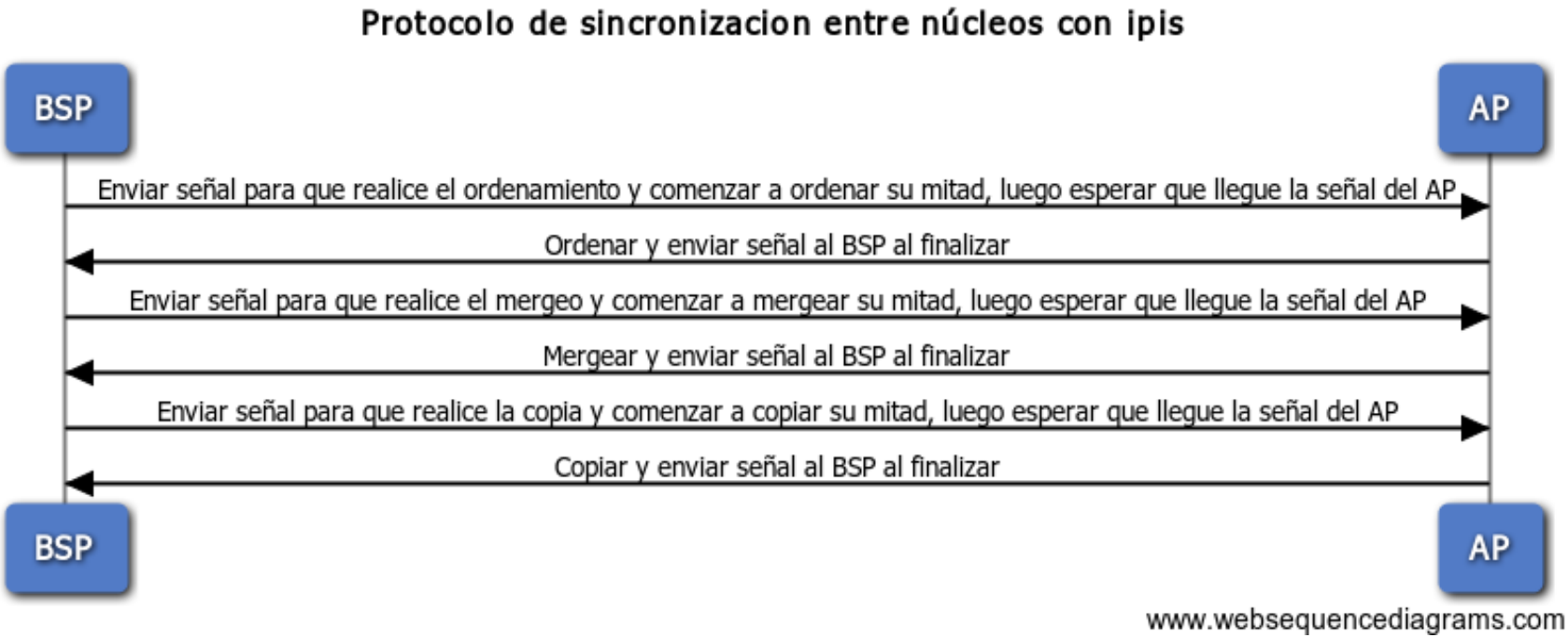
Como vimos en la sección anterior, los Application Processors comienzan su ejecución en una posición otra por debajo del primer mega en modo real, nosotros necesitamos hacer saltar la ejecución a una posición conocida por encima del mega que no se solape con estructuras del kernel ni otros módulos, la solución que propusimos es un booteo por etapas.
En este primer nivel el núcleo se encuentra en modo real, se inicializa una GDT básica en el mismo código y se salta a modo protegido, esto es necesario para poder direccionar posiciones de memoria por encima del primer mega.
Recordando secciones anteriores, cuando el BSP iniciaba el primer nivel de booteo preparaba el contexto de los APS para inicializar en niveles, en este proceso se inyecta en el código del primer nivel de booteo del AP la posicion de memoria donde esta el segundo nivel de booteo por encima del mega

Sorting Paralelo

Se optó en este caso por un algoritmo propio que combinara varios ordenamientos conocidos con el objetivo de hacer mas fácil la paralelización del experimento. En particular lo que se realiza es partir el arreglo en 2 mitades, donde cada core realiza heapsort sobre su mitad asignada y luego una intercalación de los resultados con un algoritmo también paralelo en el cual un core realiza el merge de los máximos y el otro el merge de los mínimos, luego resta copiar los resultados y concluimos. El algoritmo tiene 3 subprocesos

- Sort: Ordenamiento de mitades por cada core
- Merge: Merge paralelizado de las mitades ordenadas
- Copy: Copia del resultado a un buffer de destino

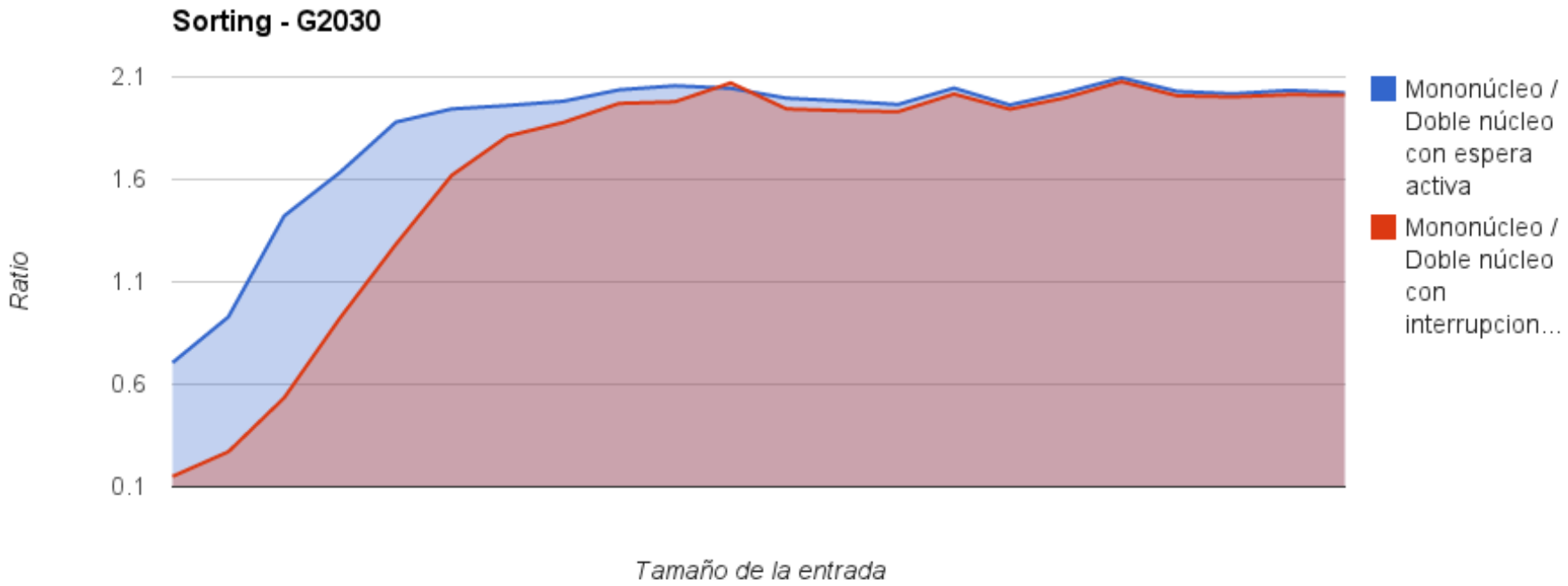

Protocolo de sincronizacion entre núcleos
AP
ap_sort
BSP
ap_finished
ap_merge
ap_cpy
polling para comenzar
Indicar al AP que comience y ordenar su mitad
Finalizar el ordenamiento de su mitad y esperar a que el AP termine su trabajo
Finalizar el ordenamiento de su mitad y encender esta bandera, luego esperar que el BSP indique comenzar el mergeo
Limpiar ap_task_finished, indicar al AP que comience el mergeo, y luego mergear su mitad
Finalizar el mergeo de su mitad y esperar a que el AP termine su trabajo
Finalizar el mergeo de su mitad y encender esta bandera, luego esperar que el BSP indique comenzar la copia
Indicar al AP que comience la copia de su mitad e iniciar la copia de su mitad
Finalizar la copia y realizar polling hasta que el AP finalice
AP
ap_sort
BSP
ap_finished
ap_merge
ap_cpy
www.websequencediagrams.com


Protocolo de sincronizacion entre núcleos con ipis
BSP
AP
Enviar señal para que realice el ordenamiento y comenzar a ordenar su mitad, luego esperar que llegue la señal del AP
Ordenar y enviar señal al BSP al finalizar
Enviar señal para que realice el mergeo y comenzar a mergear su mitad, luego esperar que llegue la señal del AP
Mergear y enviar señal al BSP al finalizar
Enviar señal para que realice la copia y comenzar a copiar su mitad, luego esperar que llegue la señal del AP
Copiar y enviar señal al BSP al finalizar
BSP
AP
www.websequencediagrams.com

Resultados

Intel® Pentium® Processor G2030 - Sorting

Elementos	Ratio Mem/Ipi	Single/DMem	Single/DIpi
2	0.213	0.705	0.150
4	0.291	0.929	0.270
8	0.375	1.422	0.534
16	0.564	1.634	0.923
32	0.682	1.879	1.283
64	0.833	1.944	1.619
128	0.923	1.96	1.810
512	0.967	2.037	1.971
131072	0.987	2.024	1.999
262144	0.991	2.095	2.077
524288	0.989	2.029	2.008
1048576	0.992	2.018	2.003
2097152	0.989	2.034	2.013
4194304	0.994	2.023	2.012


Sorting - G2030
2.1
1.6
1.1
0.6
0.1
Ratio
Tamaño de la entrada
Mononúcleo / Doble núcleo con espera activa
Mononúcleo / Doble núcleo con interrupcion...