Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico Final: DeliriOS

Organización del Computador II

Marzo 2014

Apellido y Nombre	LU	E-mail
Silvio Vileriño	106/12	svilerino@gmail.com
Ezequiel Gambaccini	xxx/xx	xxx@xxx.com

Índice

1. Introduccion

El objetivo de este trabajo práctico fue inicialmente experimentar con la arquitectura intel, realizando un microkernel de 64 bits inicializando multinúcleo. Más tarde en el desarrollo del proyecto se aprovecho para realizar algunos experimentos para analizar las mejoras de rendimiento de algoritmos que se pueden paralelizar en varios núcleos.

El informe estará dividido en secciones, cada una describiendo una parte del trabajo.

2. Desarrollo: Inicialización y contexto del sistema

2.1. Estructura de carpetas: Compilación, Linkeo y Scripts

- **ap code:** Contiene el código de los Application processors, tanto de inicialización desde modo protegido como de los algoritmos implementados.
- **ap startup code:** Contiene el código de inicializacion en modo real de los Application processors
- **bsp code:** Contiene el código de inicializacion del kernel, los algoritmos implementados, y el encendido de los Application Processors.
- **common code:** Contiene el código comun al Bootstrap Processor y los Application Processors.
- **grub init:** Contiene scripts y archivos de configuracion de grub, en la subcarpeta src se encuentra el loader que realiza el pasaje entre la máquina en especificación multiboot a modo largo de 64 bits.
- **run.sh:** Script para compilación y distribución del tp.
- **macros:** Esta carpeta contiene macros utilizadas en el código.
- **informe:** Contiene el informe del trabajo práctico en formato Latex y PDF.
- **include:** Contiene las cabeceras de las librerias.

El tp está compilado en varios archivos, de esta forma cargamos algunas partes como módulos de grub. Hay scripts encargados de compilar todo lo necesario, cada módulo tiene su Makefile y el comando make es llamado oportunamente por los scripts en caso de ser necesario.

El linkeo está realizado con linking scripts en las carpetas donde sea necesario, cada seccion esta alineada a 4k por temas de compatibilidad, asimismo hay símbolos que pueden ser leidos desde el codigo si es necesario saber la ubicación de estas secciones en memoria. Los módulos de 32 bits estan compilados en elf32 y los de 64 bits en binario plano de 64 bits, esto es por temas de compatibilidad de grub al cargar modulos.

Para correr el trabajo practico unicamente hace falta tipear $./run.sh - r$ en consola y se abrira en bochs el trabajo practico.

2.2. Integración con grub y división en módulos

Utilizamos grub para la parte del booteo para lograr un contexto inicial estable y que posibilitara la ejecución del trabajo practico desde un pendrive usb. Grub permite iniciar un kernel por medio de una especificación publicada en la web, en la que se detalla un contrato que debe cumplir tanto el kernel a iniciar como grub, entre ellas, un header que debe contener el kernel para ser identificado por grub como un kernel, y grub debe otorgarle control a dicho kernel en un estado determinado, es decir, una gdt, los registros con valores predeterminados, etc.

Esta revisión de grub no permite cargar kernels compilados en 64 bits de manera sencilla, es por esto que recurrimos a una herramienta que provee grub, que son los módulos, de esta forma podemos cargar por encima del mega distintas partes del sistema y tener claro en que parte de memoria estan cargadas.

Para solucionar el inconveniente de iniciar un kernel compilado en 64 bits, realizamos un booteo en etapas, cargando módulos y recurriendo a ellos cuando es necesario. En las próximas secciones se explicará esto con más detalle.

2.2.1. Booteo e integración con grub: Especificación multiboot, carga de modulos elf32 y binarios x64

El kernel que inicia grub debe ser un ejecutable elf32, que además cuente con un header especificado por la convencion de grub. Esto nos provee un punto de inicio del kernel en donde se sabe exactamente el estado de la máquina (`http://www.gnu.org/software/grub/manual/multiboot/multiboot.html#Machine-state`). En este primer nivel de booteo, se realizan varias comprobaciones de la especificacion multiboot, luego son inicializadas variables globales para comunicacion entre módulos y se copia el codigo inicial de los Application Processors a una posicion de memoria alineada a página de 4K por debajo del primer mega, Además son pasadas como parámetros unas estructuras de datos que contienen informacion del sistema, lo que nos interesa a nosotros en particular es una lista en donde se encuentran los módulos que fueron cargados y las posiciones de memoria en donde se encuentran copiados. Una vez obtenidas las posiciones de memoria de los módulos, realizamos un salto en la ejecución al módulo de inicializacion del Bootstrap Processor, compilado como binario plano de 64 bits.

2.2.2. Booteo e integración con grub: Mapa de memoria: Memoria baja y modulos en memoria alta

TODO: LISTA DE MODULOS Y QUE PARTE DEL TP ES CADA UNO

TODO: INSERTAR GRAFICO DE MEMORIA A LO TP3 DE ORGA2.

TODO: INSERTAR GRAFICO DE TODOS LOS MODULOS, A LO UML, INDICANDO LAS VARIABLES GLOBALES QUE COMPARTEN, IE. EL 0xABBAABBA PARA HACER EL JUMP DE LOS AP Y EL JMP AL BINARIO DE 64 BITS DESDE EL LOADER DE GRUB

2.3. Inicialización del Bootstrap processor: De estado de especificación multiboot a modo legacy x64

2.3.1. Modo Legacy x64: GDT, Paginación de los primeros 4gb

En este punto del booteo estamos en modo protegido, pero según la especificación multiboot debemos utilizar nuestra propia GDT, y lo hacemos, asignamos una GDT con 3 entradas todas de nivel 0, una comun a 32 y 64 bits de datos y 2 de código, esto es necesario para hacer los saltos entre modo real y modo protegido y modo protegido-compatibilidad x64 a modo largo x64.

TODO: IMAGEN DE LA GDT

Se utilizó un modelo de paginación en identity mapping para los primeros 64 GB de memoria. El modo de paginación elegido fue IA32e en 3 niveles con páginas de 2 megas, notar que el mapeo tuvo que sea realizado en 2 etapas, en la primera se mapearon los primeros 4gb pues desde modo protegido puedo direccionar limitado hasta 4gb y es necesario activar paginación para pasar a modo largo x64, luego ya en modo largo, completamos el esquema de paginacion a 64gb

TODO: IMAGEN DE LOS NIVELES DE PAGINACION CON SUS FLECHITAS MOSTRANDO EN ROJO QUE POR ENCIMA DE 4GB NO SE PUEDE MAPEAR

TODO: TABLAS CON DIRECCIONES DE LAS ESTRUCTURAS Y CANTIDAD DE ENTRADAS DE CADA UNA. DATOS ACA ABAJO

*Paginacion IA32e PML4 -¿ 0x140000..0x140ffff =¿ 512 entries * 8 bytes(64bits) cada una solo se instancio la primera entrada de PML4

PDPT -¿ 0x141000..0x141ffff =¿ 512 entries * 8 bytes(64bits) cada una se instanciaron las primeras 64 entradas nomas, para mapear 64gb

PDT -¿ 0x142000..0x181FFF =¿ 32768 entries * 8 bytes(64bits) cada una se instanciaron las 32768 entries para mapear 64 gb con 32768 paginas de 2 megas

Luego de establecer estas estructuras, realizamos una comprobación de disponibilidad de modo x64, y encendemos los bits del procesador para habilitar dicho modo.

2.4. Inicialización del Bootstrap processor: 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 ejecucion a un selector 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.

2.4.1. Modo Largo x64: Extensión de paginación a 64 gb

En este punto ya podemos direccionar arriba de los 4gb, entonces completamos las entradas en las estructuras de paginación para completar el mapeo hasta 64gb.

TODO: IMAGEN DEL MAPEO COMPLETO EN 3 NIVELES

2.4.2. Modo Largo x64: Inicialización del PIC - Captura de excepciones e interrupciones

Enviamos señales al PIC para programarlo de forma que atienda las interrupciones enmascarables y 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 interrupcion de reloj para sincronizacion y esperas, las demás interrupciones son ignoradas.

TODO: IMAGEN DE LA IDT Y FLECHITAS A LAS ISR

2.4.3. Modo Largo x64: Mapa de memoria del kernel

TODO: IMAGEN DEL MAPA DE MEMORIA DEL KERNEL.

2.5. Multicore: encendido de los AP's

2.5.1. TODO: Ingrese cosas de inicializacion multicore aqui.

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

Como vimos en la sección anterior, los Application Processors comienzan su ejecucion en una posicion por debajo del primer mega en modo real, nosotros necesitamos hacer saltar la ejecucion a una posicion conocida por encima del mega que no se solape con estructuras del kernel ni otros modulos, la solucion que proponemos es un booteo por etapas.

2.6.1. Booteo por niveles: Modo real a modo protegido y modo protegido en memoria alta

En este primer nivel el núcleo se encuentra en modo real, se inicializa una GDT básica 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 booteaba desde el loader de grub, inicializaba variables globales para compartir datos entre módulos, una de ellas contiene la posicion del módulo de código del Application Processor por encima del mega, es decir, que ahora solo resta que hagamos un salto a dicho módulo para continuar la ejecucion de este núcleo en memoria alta.

3. Desarrollo: Algoritmos implementados

3.1. Sorting de arreglos

3.1.1. Conjuntos de numeros pseudoaleatorios utilizados para los experimentos

3.1.2. Implementación con un unico core

3.1.3. Implementación con dos cores: Paralelización del algoritmo

3.1.4. Implementación con dos cores: Sincronización con espera activa

3.1.5. Implementación con dos cores: Sincronización con inter processor interrupts

3.2. Fast Fourier Transform

4. Resultados

4.1. Lectura e interpretación de resultados por pantalla

4.2. Resultados: Forma de medición

4.3. Resultados: Plataformas de testing, incluir fsb, ram,cache, etc

4.4. Resultados: Comparación de resultados

5. Conclusión Final