Organización del Computador II
TP3

11 de junio de 2015

Integrante	LU	Correo electrónico
Christian Cuneo	755/13	chriscuneo93@gmail.com
Julián Bayardo	850/13	julian@bayardo.com.ar
Martin Baigorria	575/14	martinbaigorria@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

El objetivo del presente trabajo practico es aprender y aplicar diferentes conceptos de *System Programming*. A partir de una implementación de un boot-sector, se programo un pequeño kernel con los diferentes mecanismos de protección y ejecución concurrente de tareas para luego poder ejecutar un juego con hasta 16 tareas concurrentes a nivel de usuario.

1.1. Inicialización

Al prender la computadora, comienza la inicializacion del `POST` (Power-On Self-Test), un programa de diagnostico de hardware que verifica que todos los dispositivos se han inicializado de manera correcta. Una vez terminado el `POST`, el `BIOS` se encarga de identificar el primer dispositivo de boooteo, ya sea un CD, un disco rígido o un diskette. En este trabajo, inicializaremos el sistema a partir de un diskette.

El `BIOS` (Basic Input-Output System) copia de memoria `RAM` los primeros 512 bytes del sector a partir de la direccion `0x7c00` de un diskette. Esto se copia comenzando en la direccion `0x1200` y luego se ejecuta el boot-sector a partir de allí. El boot-sector encuentra en el floppy el archivo `kernel.bin` , y luego lo copia en memoria a partir de la direccion `0x1200` , ejecutando a partir de la misma.

2. Kernel

El `Kernel` es una parte esencial de los sistemas operativos modernos. Se ocupa de inicializar las diferentes estructuras necesarias para utilizar las diferentes funciones del procesador.

3. Modo Real

3.1. Introduccion

Por una cuestión de compatibilidad hacia atrás, al inicializar un procesador Intel, el mismo funciona como un 8086, lo que conocemos como `Modo Real`.

En `Modo Real`, no existe la protección por hardware, por lo que cualquier código en ejecución tiene acceso a todos los segmentos de memoria y puede utilizar cualquier instrucción del 8086. Para poder utilizar otras instrucciones y funcionalidades mas avanzadas y habilitar la protección por hardware, se debe pasar a Modo Protegido.

3.2. A20

El addressing line `A20` forma parte del bus de direcciones del procesador. En un `8086`, este bus tiene 20 lineas, de la 0 a la 19. Sin embargo, cuando salio al mercado el `8086`, el primero en soportar el modo protegido, el bus de direcciones paso a tener un 24 bits. El problema que surgió es que muchos programadores en su codigo del `8086` utilizaban lo que se conoce como wrap-around. Cuando accedían a memoria, utilizaban el overflow en el bus de direcciones como parte de la lógica de sus programas. El problema que surgió entonces fue que el 80286 no soportaba este overflow, dado que tenia 4 lineas de address adicionales.

Para solucionar este problema, a IBM se le ocurrió utilizar un pin del controlador del teclado que estaba sin usar y conectarlo a la linea 20 del bus de direcciones para poder forzar el overflow en los programas viejos. Por esta razón, antes de pasar a modo protegido se debe habilitar esta linea, para poder utilizar todo el espacio direccionable por todas las lineas del bus de direcciones.

3.3. Global Descriptor Table

Antes de poder pasar a modo protegido, debemos cargar la GDT. La GDT se encarga de asignar diferentes atributos de protección a los segmentos de memoria, para luego poder habilitar la protección por hardware. Esta estructura la armamos como un array de `gdt_entry` en C.

Luego, cargamos la GDT con el comando `lgdt` y el descriptor de la GDT armado desde C.

3.4. Pasaje a Modo Protegido

Una vez armada la GDT y habilitado el A20, debemos habilitar Modo Protegido. El modo de proteccion esta definido por el bit menos significativo del registro `RCR0` . Usando un & logico, habilitamos este bit.

Una vez que tenemos todas las estructuras necesarias armadas, hay que hacer un `jump far` a alguno de los segmentos que definimos en la GDT. De esta forma finalmente habilitamos la proteccion por hardware y pasamos a Modo Protegido.

4. Modo Protegido

4.1. Memory Management Unit

4.2. Task State Segment

4.3. Scheduler & Interrupt Descriptor Table