Informe

Arnix

Modo protegido con GRUB

31 de mayo de 2011

Autores:

Axel Wassington	Legajo: *50124*
Horacio Miguel Gomez	Legajo: *50825*
Tomás Mehdi	Legajo: *51014*

Índice

1. Decisiones e implementación del sistema

1.1. Código

Se utilizo un mix de assembler con C por decisión de la catedra. Es muy importante aclarar este punto, por que varias instrucciones solo pueden hacerse desde assembler y para simplificar la codificacion del TP se usa C llamando a assembler o assembler llamando a C.

1.2. Compilación, linkedición y ejecución

Reemplazamos el compila por un makefile y el arma por un build. El build puede llamarse con el parametro clean o no. Si se llama con el parametro se hace un make clean y make. Sino solo se hace el make. Luego de estos en los dos casos si el código compila se abre el bochs.

1.3. Pantalla

Se utilizaron ANSI escape squences, estos son chars incrustados en el texto que se utiliza para controlar el formato, color y otras opciones de salida en las terminales de video en formato texto. Nosotros implementamos los siguientes ANSI scape characters:

Esc[2J Borra la pantalla y mueve el cursor a (line 0, column 0)
Esc[#;#;...m Cambia el modo de graficos segun los siguientes atributos:

1. Text attributes
   - 0 All attributes off
   - 1 Bold on
   - 4 Underscore (on monochrome display adapter only)
   - 5 Blink on
2. Foreground colors
   - 30 Black
   - 31 Red
   - 32 Green
   - 33 Yellow
   - 34 Blue
   - 35 Magenta
   - 36 Cyan
   - 37 White

3. Background colors
    40 Black
    41 Red
    42 Green
    43 Yellow
    44 Blue
    45 Magenta
    46 Cyan
    47 White

1.4. Interrupciones

Para manejar las interrupciones creamos dos estructuras, una de ellas describe una interrupt gate y la otra tiene un puntero a un array de interrupts handlers. Tambien creamos una estructura que esta integrada por todos los registros del micro, la que nos permite obtener los datos necesarios para saber como ejecutar las distintas interrupciones. Las INT80h es una interrupcion importante, mediante la cual se hace un paso desde el userspace al kernel space para utilizar algunos codigós. Mas adelante en este informe se detalla el uso del a INT80h. También es importante mencionar que remapeamos todas las interrupciones para que no se pisen con las exepciones(posisiones 0-31 de la idt) lo que nos facilito mucho la programacion.

1.5. Shell

Al iniciar la shell se agregan todas los comandos, esto se hace utilizando una estructura que tiene un puntero a función y el nombre del comando. El maximo tamaño de un comando son 1000 caracteres, si te pasas ignora los las siguitenes entradas. Cada 500 caracteres se hace un flush del stream y no te permite borrar los 500 caracteres anteriores. Se puede ingresar un argumento entre comillas para que lo considere como un solo argumento. A diferencia de las shells que normalmente utilizamos (Unix/Linux), nuestro shell ignora los espacios del principio y del final y los espacios multiples entre medio de distintos argumentos.

1.6. Division del Kernel y user space

Hicimos una muy buena implementacio para que la division de estas dos partes sea notoria. Estos se logró dividiendo en diferentes

carpetas los distintos archivos relacionados con cada espacio. Todas las llamas a sistema se hicieron a travez de funciones codificadas en systemcalls.asm.

1.7. Funcionamiento del comando getCPUspeed

Muestra la frecuencia de trabajo del CPU, usando la funcion RDTSC(Read Time-Stamp Counter) de assembler la cual retorna la cantidad de instrucciones realizada hasta el momento desde el inicio del procesador y el PIT(Programable Interval Timer). El PIT es un periferico conectado a la IRQ0 del master PIC. Utilizando estos elementos podemos obtener una cantidad de instrucciones en un intervalo de tiempo. La forma de hacerlo es pidiendo un RDTSC, dejar pasar un tiempo fijo y volver a pedir un RDTSC. El tiempo fijo lo generamos con una cantidad coherente de timer ticks, para ello no debe ser muy pequeña. Ya teniendo estos datos solo falta hacer algunas cuentas que nos devolveran la velocidad del CPU en MHz. Dichas cuentas son para hacer el calculo mas presiso, hacer la resta entre los 2 valores de RDTSC en orden de obtencion(el primero menos el segundo) multiplicado por 18(cantidad de ticks para alcanzar 1 segundo) dividido la cantidad de ticks que utilizamos y a eso sumado la resta en orden de obtencion de los RDTSC dividido la cantidad de ticks multiplicado por 5 que representa el .2 de los 18.2 ticks que hay en un segundo. Esta cuenta nos devuelve la cantidad de instrucciones por segundo. Este valor dividido $1024 * 1024$ nos da la velocidad en MHz de CPU.

1.8. INT80h

Similar a la INT80h de Unix/Linux; la INT80h de Arnix segun el valor en el registro EAX elige una instruccion.Las instrucciones que puede realizar son las siguientes:

1. Con el valor 3 en EAX hace un *read* usando los valores de EBX, ECX y EDX. En estos regristros debe estar el tamaño de lo que se va a leer, el source buffer y un file descriptor(de donde leer) respectivamente.

2. Con el valor 4 en EAX hace un *write* usando los valores de EBX, ECX y EDX. En estos regristros debe estar el tamaño de lo que se va a escribir, el source buffer y un file descriptor(donde esribir) respectivamente.

3. Con el valor 5 en EAX hace un *cpuspeed* guardando la cantidad de IPS(instrucciones por segundo) en donde apunta el registro

EBX.

1.9. Printf

Se codeo este comando lo mas parecido al de la libreria estandar de C, pero a pesar de eso no todas las opciones estan en ella. Las opciones del printf que se hicieron son las más utilizadas; imprimir un $int, string, decimalaoctal, decimalahexa, uncaracter, unsignedint$. El printf utiliza putchar.c la cual utiliza la funcion __write.

1.10. Teclado

Se crearon dos mapeos de teclado, uno es para cuando el shift esta apretado y el otro cuando no. Hay cuatro vectores de listeners que se utilizan para los distintos estados del teclado según el keypress de las teclas $CTRL$ y ALT. Cuando se aprieta el $BLOCKMAYUS$ se activa una variable, mediante la cual el teclado actua simplemente sumando un ascii para que los valores cambien de lowerCase a upperCase o de upperCase a lowerCase. Al apretar a la vez $CTRL + ALT + DEL$ el sistema se reinicia, mandando un valor determinado a cierta dirección del teclado.

1.11. IN-OUT

Tenemos un vector de diez lugares en los cuales estan los distintos tipos de $in-out$. En la primera posición del vector ($index = 0$) esta el standard in y en la segunda posicióm el standard out

2. Referencias

Esta sección detalla las distintas fuentes de información utilizadas para el desarrollo del TP Especial. Es importante destacar que son las mismas fuentes enviadas en un mail previo a la entrega.

2.1. Interrupciones

El manejo de interrupciones es similar al usado en el siguiente tutorial:

http://www.jamesmolloy.co.uk/tutorial_html/4.-The%20GDT%20and%20IDT.html

Nos pareció interesante la opción de crear un wrapper para las idt y luego desde C simplemente asignar handlers a las convenientes, un wrapper se encarga de que a C le lleguen los parametros.También creimos importante tener las entrys para las primeras 31 exceptions del procesador, para evitar resets si por ejemplo dividimos por cero.

2.2. Pantalla

Nos basamos en la implementación de Linux, la pantalla puede recibir scape chars para limpiar la pantalla, o imprimir en colores. Esto era conveniente debido a que al utilizar la int80, no necesitamos parametros extra, para estas funcionalidades extra.

2.3. Reboot

Luego de probar soluciones sucias, como hacer que el procesador triple-faultee y se reinicie la pc. Encontrámos la solucion de enviar la señal de reset desde el controlador de teclado en:

http://wiki.osdev.org/Reboot

2.4. Paginas web utilizadas

En general leimos mucho de:

http://wiki.osdev.org/Main_Page (y sus foros)
http://www.jamesmolloy.co.uk/tutorial_html/index.html
http://www.osdever.net/tutorials/view/brans-kernel-development-tutorial