Arquitectura de las Computadoras

2do cuatrimestre - 2013

Trabajo Práctico Especial:

Modo protegido con GRUB

Integrantes:

- Cristian Ontivero - 51102
- Valeria Serber - 51021

Fecha de entrega: 12-11-2013

Introducción

El trabajo práctico especial consistió en la realización de un sistema booteable para una computadora de arquitectura de 32bits, que muestre algunas de las características del modo protegido disponible en los microprocesadores Intel, y que utilice recursos de hardware.

El presente informe tiene como finalidad explicar algunas de las decisiones tomadas en la realización del sistema booteable con respecto al diseño y funcionalidad.

Funcionalidad

División entre Kernel y User space

Para hacer una separación entre User Space y Kernel Space, se decidió implementar una interrupción basada en la int 80h de Linux, de modo que sólo un número limitado de funciones tuvieran acceso directo y conocimiento de implementaciones específicas, direcciones de memoria, y variables de bajo nivel dependientes de la arquitectura. Ejemplo de éstas son el write y read del kernel space. Gracias a esto se logra una mayor modularización del código del kernel, logrando portabilidad, ya que con tan solo reescribir las funciones de más bajo nivel (las de assembler, que dependen de la arquitectura en la que uno trabaja), el resto del código se puede mantener idéntico y compilar para otras arquitecturas.

Esta decisión sirvió también, para que un programa, por ejemplo el Shell, le pida al Kernel utilizar alguna función de otro driver, por ejemplo, el de video. De esta manera, los drivers no se comunican directamente entre sí, sino que lo hacen mediante el Kernel.

La rutina de atención de la int 80h, escrita en assembler, se ocupa de cargar en el stack los registros con los valores correctos, y luego llama a un handler en C, que se encarga de llamar a la función del Kernel que corresponde, basándose en un valor numérico que identifica a cada función (emulando la idea de sys_calls utilizada en varios sistemas operativos modernos).

Debido a que cada función recibe diferentes parámetros, se los declara en C de tipo dword (typedef de unsigned int) para denotar que ocupan 4 bytes, y luego se los castea al tipo que corresponda a cada sys_call.

Funciones de librería estándar de C

La implementación de la int 80h permitió que la implementación de getc y putc sea muy simple. Solamente se llama a la función read y write del user space, respectivamente, con los parámetros adecuados. Si bien nos hubiera gustado quedarnos con la definición original de la librería de C de

estas funciones, las cuales reciben un FILE * stream, no pudimos hacerlo dado que hubiéramos necesitado implementar malloc, y la mayoría de los parámetros de la estructura FILE resultaban innecesarios para la magnitud del sistema implementado (entre otras cosas, sin contar con un file system). En estos casos, se hizo uso de un int para el file descriptor, que de todas formas era lo único necesario dentro de la estructura FILE a la hora de llamar a read y write.

Las macros getchar y putchar, están implementadas de modo que llamen a getc y putc, con STDIN y STDOUT respectivamente. Las funciones printf y scanf, de igual manera, hacen uso de putc y getc respectivamente. Se decidió implementar también un "uprintf", que imprime en la pantalla superior requerida en el enunciado. Debido a la similitud entre ambas, se optó por implementar vfprintf, y que ambas deleguen a esta la responsabilidad de impresión llamándola con los parámetros adecuados.

Video

El driver de video contiene las funciones que acceden directamente a la zona de memoria donde se encuentra la pantalla, y modifican el contenido.

La pantalla está separada en dos partes. La superior, de 10 filas y 80 columnas, tiene el propósito de mostrar los valores de los registros al presionar Ctrl+R. La inferior, de 15 filas y 80 columnas, es la salida estándar del resto del sistema. Para saber en qué lugar de la pantalla se debe escribir, se decidió tener dos variables, una para cada pantalla, que guarden ese dato. Esas variables sólo son modificadas y accedidas por el driver de video.

Teclado

Se implementó la interrupción 09h para el teclado, que se recibe mediante la IRQ1 del PIC. La rutina de atención escrita en assembler, accede al puerto de entrada/salida 60h, de donde toma un scancode, que puede ser un makecode o breakcode, según si se está presionando o soltando la letra. Posteriormente se llama a una función que analiza dicho scancode.

Los breakcodes son mayoritariamente ignorados, con excepción de unos pocos relevantes, que son el de las teclas shift y control. En el caso de recibir un makecode, éste se transforma a código ascii utilizando unas tablas, y luego se almacena en el buffer del teclado. Se optó por un buffer de teclado circular, con dos índices que controlan donde insertar y de donde retirar el próximo caracter, y se deja un espacio vacío para diferenciar los casos de buffer lleno y vacío. En él se almacena todo el historial de teclas presionadas, hasta que no se puedan insertar más, o se vacíe por ser consumido por alguna función que termine llamando a read con el file descriptor del stdin.

Para conocer el estado presente del teclado, se cuenta con flags para saber si el caps locks está activado, y si el shift o el control se encuentran presionados.

Shell

Se decidió que el shell cuente con su propio buffer y que su contenido se corresponda con lo que se

muestra en pantalla cuando esta aplicación está corriendo. Por ejemplo, allí no se almacenan backspaces, sino que se ejecuta la acción de borrar la tecla anterior y dejar el buffer igual que como se ve en pantalla. Esto es así para facilitar la tarea de parsear el contenido en busca de un comando para ejecutar, además de ser lo más intuitivo y utilizado en sistemas modernos. Si el buffer se llena, en pantalla se observa que no se puede seguir agregando caracteres, pero sí se puede borrar o presionar enter para parsear lo escrito hasta el momento.

En una primera instancia, dejábamos que se pueda seguir ingresando teclas y viéndolas en pantalla pero sin agregarlas al buffer, de modo que en realidad se estaban ignorando a la hora de parsear. Pero debido a que esto puede resultar confuso para el usuario, se mantuvo la correlación visual con el contenido del buffer, y se impide la escritura una vez llenado el buffer interno.

Registros

En cualquier momento de la ejecución del sistema, al presionar Ctrl+R se muestra en la pantalla superior el estado de los registros en el momento de presionar las teclas. Se presentó la incógnita sobre cuál era el momento correcto para guardar el estado de los registros. Se decidió que la función encargada de guardar esa información debía ser la rutina de atención de teclado, por lo que los guarda al comienzo de la rutina, y luego, más adelante, si se detecta que se presionaron las teclas mencionadas, se procede a imprimir en pantalla lo almacenado.

La información que se muestra es: los registros de 32 bits, los flags y los registros de segmento de 16 bits. La función que guarda estos datos está escrita en assembler. Primero reserva espacio para guardar la información. Luego pushea en el stack los registros de segmento ss, ds, es, fs, gs; y los registros de 32 bits, eax, ebx, ecx, edx, esp, ebp, esi, edi. Tanto los flags como el registro de segmento de código cs, y el eip, ya estaban pusheados en la pila ya que se encuentra adentro de una rutina de interrupción. Por lo tanto para acceder a todos estos valores, basta con acceder a la pila secuencialmente y guardar esos datos en el espacio que habíamos reservado. Luego se imprime accediendo a ese vector que quedo guardado.

CD-ROM

Se requería tres funcionalidades distintas con respecto al CD ROM: abrir la lectora, cerrarla, e imprimir la información sobre la capacidad de la misma. Resultó ser más dificil de lo que esperabamos, habiendo falta de buena documentación sobre como realizar los procedimientos pedido. A ultimo momento fuimos capaces de abrir y cerrar la lectora, pero seguimos sin lograr implementar el infoCD. Además, resulto ser extremadamente sensible a cambios en el codigo, y aun en veces que creimos no haber modificado lo ya implementado y simplemente tratado de agregar mayor funcionalidad (por ejemplo, la adición de polling a los IDE primario, secundario, y al master y slave), volvia a no funcionar y nos vimos forzados a volver a versiones pasadas del codigo.
En el presente trabajo, funciona siempre y cuando el CD-ROM se halle en el Primary IDE slave.

Conclusiones

- Nos pareció muy buena la decisión de implementar la int80h, ya que además de aportar un buen diseño, hizo muy simple implementar el resto del sistema debido a que write y read centralizaron todo el funcionamiento.
- Para implementar el control+R hubo que buscar los valores de los registros en el stack para poder acceder a los valores del instante correcto de tiempo. Esto fue interesante para aprender como se pushean los registros en el stack, y cómo al recibir una interrupción, se pushean los flags, eip y cs, razón por la cual es necesario utilizar la instrucción iret para activar las interrupciones.
- Sobre el CD, sólo logramos que abra y cierre la disquetera siempre y cuando ésta se encuentre en el Primary IDE slave.