Trabajo Práctico Especial

Teresa di Tada y Juan Pablo Rey

November 19, 2013

Objetivo

Realizar un sistema booteable por una PC que muestre algunas de las características del Modo protegido de los microprocesadores de Intel y utilice recursos de hardware.

Consideraciones de diseño

En concordancia a lo aprendido hasta ahora en la materia, se mantuvo un diseño en capas; enfatizando sobre todo la separación entre kernel y user space siempre que fuera posible.

Se implementaron system calls estándar a la usanza Linux: se implementó la $INT80h$ como único modo de comunicación con el kernel y se implementaron las llamadas a sistema $\_\_write$ y $\_\_read$, las cuales mantienen la misma especificación que sus homólogas de Linux (sys_write y sys_read).

Cabe aclarar que, al no contar ni con un filesystem ni con alocación de memoria ni con un scheduler por cuestiones de simplicidad y no atinencia a los contenidos de la materia, no siempre fue posible atenerse a las convenciones de Linux y/o C, pero se hizo lo posible para minimizar la distorsión.

Se emularon entonces la entrada y salida estándar, junto con una salida especial para la pantalla de los registros (denominada "REG_OUT"). De esta forma se mantuvo el código lo más modular posible, permitiendo mantener la prolijidad al mismo tiempo que la posible extensión y el fácil mantenimiento del sistema a futuro.

La única anomalía de diseño fue entonces la relacionada a la impresión de los registros y descriptores del microprocesador siempre que se presionara $Ctrl+R$. Puesto que se requería que esta pudiera gatillarse en cualquier momento de la interacción entre el usuario con la PC, se decidió junto con la cátedra que lo mas sencillo era que, por un lado, el driver de teclado se encargara de manejar esa "interrupción", llamando él mismo a las funciones de impresión cuando correspondiera (lo cual viola el concepto modular del sistema por permitir a un driver el comunicarse por su cuenta con capas de nivel más alto); por otra parte, las rutinas de ASM fueron modificadas para que guardaran todos sus parámetros.

Implementación

Se detallan a continuación los módulos más relevantes del sistema operativo:

Kernel

Componente principal del SO, es la que se encarga de inicializar el sistema, de mantener corriendo el shell contiuamente para que el usuario pueda usarlo indefinidamente y, lo más importante, contiene las funciones $\_\_read()$ y $\_\_write()$, las cuales permiten abstraer al sistema operativo en sí mismo de la arquitectura, lo cual le brinda portabilidad y robustez.

Inthandlers

Contiene todas las rutinas manejadoras de interrupción, entre ellas la del timer tick (la cual no hace nada, puesto que, como se mencionó previamente, no hay un scheduler), la del teclado y la $INT80h$.

Video

Contiene todo lo relacionado al manejo de video, el cual se manejó como si fueran dos displays separados ($STD\_DISPLAY$ y $REG\_DISPLAY$).

Las funciones que se exponen de esta librería solo necesitan del identificador del display para poder saber todas las restricciones de impresión necesarias (especialmente las relacionadas al tamaño del display virtual en cuestión).

Por otra parte, se decidió colorear los displays para marcar su diferencia sin perder espacio de pantalla, coloreando solo una vez cada display al inicializar la librería y no tocando las celdas relacionadas al color durante el resto de la ejecución del S.O..

Cabe aclarar que si bien ambos displays comparten mucha de la misma funcionalidad (como el scrolleo automático una vez que se intenta escribir más allá de la última línea y el soporte de caracteres especiales como la tabulación y el enter), se especializó cada uno para un uso particular (que será el que luego les dará el sistema operativo).

El $STD\_DISPLAY$ se pensó de manera tal que permitiera todas las funcionalidades esperables en una salida estándar de un sistema operativo básico, moviendo el cursor a la posición siguiente al último caracter escrito e implementando la posibilidad del borrado de caracteres. Esto último se logró a través del uso de un buffer circular de video, el cual guarda la posición previa a escribir cada caracter, permitiendo entonces el borrado simple de caracteres especiales como la tabulación y el enter.

Mientras tanto, se decidió no implementar la funcionalidad de borrado de caracteres en el $REG\_DISPLAY$, puesto que no se le vio utilidad a esta funcionalidad en una pantalla especializada para la impresión automatizada de registros

Keyboard

Contiene el driver de teclado; para el mismo se utilizó un buffer circular, el cual permite no tener que refrescarlo ni cambiar su tamaño y utilizar la técnica de polling desarrollada previamente en clase. Como se dijo previamente, por los requerimientos del S.O., el driver de teclado también se encarga de "escuchar" la combinación de teclas $Ctrl + R$ y de "manejarla" llamando a las funciones de impresión correspondientes.

En cuanto a las funcionalidades que se soportan encontramos: el *Caps Lock* para utilizar letras mayúsculas, la combinación de éste con Shift diferenciando la impresión de letras minúsculas y símbolos, la funcionalidad del Shift, que se presionen las teclas especiales por más de un make code, la combinación $Ctrl + R$, la combinación de $Ctrl$ y cualquier otra tecla que no imprima nada, y la opción de presionar $Ctrl + Shift + R$ sin que se impriman registros, al igual que cualquier otra combinación de $Ctrl + R$ que tenga teclas que no sean las ya mencionadas.

Esto último se logró utilizando un contador de cantidad de teclas presionadas en el driver de teclado al recibir los make codes y break codes, lo que podría servir en el futuro para agregar alguna otra combiandción de teclas que lanze un comportamiento especial.

Se decidió utilizar el teclado en inglés para la matriz que mapea scancodes y asciis por facilidad, ya que hay signos en español en la tabla de ascii extendido y no en la común.

Por otro lado, no se sporta el $Ctrl$ derecho ni el $AltGr$ -éste último envia un make code y dos break codes y hay que considerarlo aparte para contar la cantidad de teclas-, ni las funcionalidades especiales del resto del resto de las teclas especiales no mencionadas en las funcionalidades soportadas -$f1$,Del,Esc,etc-.

Shell

Contiene un shell sencillo el cual da la bienvenida al usuario al iniciar el S.O. para luego encargarse de interpretar los comandos que ingrese el usuario y ejecutarlos. Esto se hace en primera instancia mediante un buffer propio, el cual permite un mejor manejo de los caracteres que se ingresen y permitiendo el pasaje de múltiples parámetros a los comandos sin necesidad de alocar memoria.

No se permite una navegación compleja como en las terminales actuales (no se soportan las flechas ni tampoco combinaciones del estilo de $Shift + Enter$; sí se soporta el overflow).

Se decidió que el comando por defecto sea '*echo*'.

Commands

Contiene todos los comandos adicionales del shell, desde 'echo' hasta las funciones de prueba requeridas por la cátedra. Se decidió que todos los comandos recibieran una cantidad "variable" de argumentos de tipo $char*$ (con un máximo

de $SHELL\_MAX\_ARGS$, dado que no se puede alocar memoria) y un entero indicando cuántos argumentos fueron recibidos.

Librería estándar de C

Se implementaron las funciones *putc*, $printf$, *getc* y $scanf$ de la librería estándar de C con las salvedades previamente mencionadas (nacidas mayoritariamente del no tener un file system). Solo ellas acceden a $\_\_write()$ y $\_\_read()$, manteniendo la modularidad y el diseño en capas.

Para poder tener argumentos variables, y por la complejidad de implementación de los mismos de una manera no dependiente de la plataforma, se decidió utilizar la implementación GNU de $< stdarg.h >$. Esta, por estar enteramente basada en macros, es independiente de la plataforma para la que se desee compilar el S.O. a futuro.

También se implementó la función rprintf, la cual es idéntica a printf pero que imprime a REG_OUT en vez de STD_OUT.

Comandos ATAPI para el CD

Se implementaron comandos con la interfaz ATAPI para abrir, cerrar y leer la capacidad de un CD. No se logró hacer éstó último, sí los dos primeros. El código de los mismos está en Assembler a pesar de que hubiera sido más sencillo implementarlo en lenguaje C utilizando inline assembler para inb, oub y otras instrucciones necesarias.

Se buscó utilizar el comando identify de la interfaz ATA para reconocer dónde hay un CD, ya que estos no son compatibles con ATA y a su vez devuelven bits útiles para identificarlos. También se buscó utilizar el Soft reset de ATA.

Info Registers (Ctrl+R)

El info registers muestra los registros al momento que se presiona esta combiación de teclas. Para que funcione en cualquier programa y existan en el momento indicado los datos necesarios, se lo implementó desde el driver de teclado, guardando siempre los registros de manera que el driver reciba siempre los registros y los tenga disponibles al necesitarlos.

Los flags, eip y cs se buscaron el el stack ya que la interrupción los coloca allí con 32 bits cada uno. Se tuvo cuidado de recibir en la función de C del driver de teclado los registros con su tamaño real: si son de 16 bits como el code segment, se reciben como unsigned short int en cambio si son de 32 como int.

Se realizó un comando que testeara el cambio de *eax*, *ebx* y *edx* a través de un loop sobre *ecx* para comprobar el correcto funcionamiento del Info Registers.

Manual de uso

Se soportan los siguientes comandos:

echo	Símil al echo de $UNIX$, toma uno o múltiples parámetros y los imprime en salida estándar
help	Imprime una lista de ayuda por salida estándar
logo	Imprime el logo del SO por salida estándar
putc	Test simple de putc para la cátedra
getc	Test simple de getc para la cátedra
printf	Test de printf para la cátedra
scanf	Test de scanf para la cátedra
regs	Test del sistema de info registers ($Ctrl + R$)
abrirCD	Abre la lectora de CD
cerrarCD	Cierra la lectora de CD
infoCD	Muestra por salida estándar el tamaño del CD dentro de la unidad

Consideraciones

Se toma a los espacios y tabulaciones entre palabras solo como separadores, por lo que, por ejemplo, los llamados siguientes serán equivalentes entre sí:

```
echo hola mundo
echo            hola      mundo
        echo  hola mundo
```