Arquitectura de Computadoras

Trabajo Práctico Especial

Integrantes:

- de la Puerta Echeverría, María
- Elli, Federico

Fecha de entrega: 13/11/2012

Introducción

Para comenzar el trabajo fue necesario comprender el funcionamiento del GRUB, encargado de entregarnos el control con un modelo flat de memoria e inicializar el procesador y sus tablas; además del resto de los programas que permiten en instancias posteriores probar todo lo programado en un entorno virtual (Mtools, Bochs).

Trabajamos con la versión 10.04 de Ubuntu, versión en la cual los programas mencionados funcionan sin problemas. Contando entoñ bvnces con un entorno aceptable para poder comenzar a testear lo que ya teníamos programado en C.

Desarrollo y decisiones de diseño

Contando con un entorno completo, Bochs, Eclipse, Java, gcc, nasm, vim, y otros programas, comenzamos a desarrollar.

Basándonos en lo programado el primer cuatrimestre de 2012, se redujo la dificultad de comenzar el trabajo ya que nos vimos familiarizados con los primeros pasos a realizar. Dada la base sobre la cual deberíamos empezar, comenzamos por habilitar la interrupción de teclado y lograr poder escribir en pantalla. No realizamos cambios sobre la forma en la que decidimos manejar la interrupción de teclado y la disposición del teclado, y más adelante discutiremos este aspecto.

En cuanto a la lectura y la escritura, decidimos que las funciones primitivas read y write funcionarían a través de la interrupción de software INT 80h. Para este fin, desarrollamos el handler de la interrupción en código de assembler. Por lo tanto, en la IDT insertamos tres interrupciones principales, que son las que usaríamos para el funcionamiento del sistema, la INT 80h, INT 09h (teclado) e INT 08h (timer tick). Luego para agregar la funcionalidad de y agregar interrupciones, creamos un handler de interrupción genérico.

Con gran parte del funcionamiento realizado, nos concentramos en crear algunos casos de prueba de las funciones de la librería, como *printf*, y a desarrollar el *malloc* y el *free.* Las decisiones de diseño sobre estas funciones se charlarán más adelante.

A continuación, se explica en detalle algunas secciones en las que decidimos hacer hincapié a la hora de comentar las decisiones de diseño.

Shell

La consola se maneja mediante un vector de 25 por 80 que representa el espacio visible en el monitor. Utilizamos un buffer sobre el cual escribimos y luego se vuelca sobre la dirección.

Fue necesario aprender sobre los atributos con los que pueden mostrarse los caracteres, para poder así imprimir las letras que el teclado nos brinda mediante *scancodes*. En la imagen se puede ver como funciona..

Teclado

El teclado fue simple de implementar, una matriz de 60x2 caracteres. Como decidimos disponer de dos tipos de teclado, inglés y español, el *default* es en inglés. Decidimos usar una matriz para representar los caracteres en su disposición contigua y en su disposición con SHIFT.

Mediante la lectura del puerto 60h una vez generada una interrupción del teclado, se obtienen los *scancodes* que se generan al presionar o soltar una tecla. Mediante el uso de *flags*, se manejó el uso de teclas especiales, como SHIFT, ALT y CTRL.

Funciones de la biblioteca standard de C

Para implementar estas funciones, usamos el libro “C Programming Language” de Kernighan y Ritchie como guía, ya que el enunciado pedía que fueran lo más cercano posible a estas.

Malloc

Optamos por una implementación propia de *malloc*, la funcionalidad que ofrece es la misma que en la implementación de K&R y cumple con los requisitos pedidos en el enunciado.

Trabaja con bloques de memoria de distinto tamaño. Un bloque consiste en la posición de memoria donde empieza, el tamaño en bytes y un *flag* para indicar si el bloque se encuentra libre u ocupado.

Al realizarse las peticiones de memoria mediante llamadas a *malloc*, el vector se va llenando de los bloques que se van creando, así hasta que no hay más memoria libre en el sistema.

Free

La función *free*, libera los bloques de memoria que se fueron pidiendo. Si se libera el último bloque que se pidió la liberación es real, se borra el bloque de memoria.

Si se libera un bloque que no sea el último que se pidió lo que hace es cambiar el *flag* e indicar que el bloque ahora está libre. dejando un bloque libre entre otros bloques ocupados. En esta caso, el *malloc* puede reutilizar este bloque si en el futuro se pide un bloque del mismo tamaño del que quedó marcado como libre.

Funcionalidad

Teclado

Brindamos 2 configuraciones de teclado funcionales, en español e inglés, con mayúsculas y tildes. Para cambiar la disposición del teclado, se realiza a través de un comando ***language*** seguido del parámetro ***EN*** o ***ES*** según qué teclado se desee.

Multi Terminales

Aprovechando la funcionalidad agregada al TP Especial del cuatrimestre pasado, decidimos agregar la opción de multi terminales. Para navegar por las distintas terminales, se deben cambiar combinando la tecla ***alt + [num]***, donde num, es un número del 1 al 9.

Help Desk

Al ejecutar el comando ***help*** se muestra una lista de comandos en pantalla. Además del nombre del comando, y sus posibles argumentos, se ofrece una corta descripción de la función del mismo.

IDT

Como consigna del trabajo práctico especial, se pidió implementar un comando que mostrara la información de la IDT (Interrupt Descriptor Table). Además, se debían crean comandos que permitieran agregar y quitar elementos de la tabla.

Para mostrar la tabla, se usa el comando ***idt*** sin parámetros. Esta muestra el número de la interrupción en hexadecimal, el higher y el lower offset y el selector.

Luego, para poder agregar y quitar elementos, se utiliza el mismo comando seguido de los parametros ***add***, para agregar interrupciones o ***del***, para eliminar interrupciones. Ambos deben estar seguidos del número de la interrupcion. Ejemplo: idt add 7 (Agrega la INT 7h).

Malloc

Otro punto pedido en el trabajo fue la implementación de la función malloc para asignar un bloque de memoria y dar al usuario el puntero al comienzo del mismo. Para este fin, el usuario debe ingresar el comando ***malloc [arg]***, donde arg es la cantidad de memoria en bytes que se quiere reservar. Al ejecutar el comando, si se pudo reservar la cantidad de bytes pedida, se muestra en consola el puntero al usuario.

Free

Para liberar la memoria asignada previamente con malloc, se utiliza el comando f**ree [arg]**, donde arg es la dirección de memoria que se desea liberar.

Mem

Se agregó el comando mem, para que el usuario pueda ver los bloques de memoria que ya fueron asignados alguna vez y su tamaño junto con flag *free*, que indica si el bloque se encuentra ocupado o no.

Funciones de prueba

Para probar alguna de las funciones de nuestro código, agregamos un par de comandos que el usuario puede ejecutar y ver su comportamiento. Al ejecutar el comando ***test***, se puede ver una pequeña demostración del *printf* y sus distintas formas de impresión.

El comando ***echo [args]*** imprime el contenido de args en pantalla y el comando ***prime [args]*** imprime si el número ingresado es o no primo.

Conclusiones

Concluimos que, al realizar este trabajo nuevamente, comprendimos mejor el funcionamiento del sistema. Nos pareció interesante implementar la función malloc y asignar bloques de memoria de distintos tamaños y luego poder liberarlos y mantener varios bloques predefinidos que luego se pueden reasignar. Esto nos sirvió para comprender un poco más cómo funciona la memoria.

Nos manejamos muy bien con las interrupciones ya que nos resultó sencillo implementar las funciones de la IDT, ya sea para visualizar los elementos del a tabla o agregar o quitar interrupciones. Aunque no haya sido especificado por el enunciado, la decisión de utilizar la INT 80h fue buena ya que nos permitió entender mejor el funcionamiento del sistema operativo y las diferentes *capas* que existen entre el hardware y el software y cómo sólo las funciones primitivas son las únicas que deben comuncarse diréctamente con el hardware.