Informe

Integrantes:

- Hirschowitz, Kevin 52539
- Rivera Ignacio 53029

Profesores:

- Merovich, Horacio Víctor
- Vallés, Santiago Raúl
- Nizzo Mc Intosh, Augusto

Materia:

- Arquitectura de computadoras

Implementaciones y decisiones de diseño

Biblioteca estándar:

Funciones de argumentos variables

A la hora de programar funciones de la biblioteca estándar tales como printf y scanf, fue necesario implementar el concepto de cantidad de argumentos variables. Implementamos las mismas siguiendo el modelo que propone el libro “El lenguaje de programación c” de Kernighan y Ritchie y utilizando el header stdarg.h de Internet el cual nos proporciono macros tales como:

- void va_start(va_list *ap*, *argN*);
- void va_copy(va_list *dest*, va_list *src*);
- type va_arg(va_list *ap*, *type*);
- void va_end(va_list *ap*);

Interrupciones:

Ademas de la implementación de la int 09 e int 08, de teclado y timer tick respectivamente, implementamos la interrupción 80. La cual utiliza el valor “pusheado” eax como syscall para elegir entre read o write y luego pasando registros ebx y ecx como parámetros para las mismas. Es importante la implementación de esta interrupción para la correcta separación del user space respecto del kernel space.

Teclado:

Decidimos utilizar un arreglo el cual tiene los caracteres ascii en correspondencia con los scan codes obtenidos por la interrupción de teclado. También implementamos un segundo arreglo de caracteres ascii el cual contiene los caracteres en mayúscula ya que implementamos en el teclado el funcionamiento de Bloq Mayus, shift derecho y shift izquierdo.

Asimismo también logramos que mediante la combinación de caracteres(control + r), ya sea control izquierdo o derecho se actualicen en pantalla el valor de los registros:

- Eax
- Ebx
- Ecx
- Edx
- Esi
- Edi
- Eflags

Esto se logro realizando un guardado en el stack de todos estos registros cada vez que se realizaba una interrupción de teclado.

Buffer:

Nuestra implementación de buffer se basa en un vector el cual cuenta con 2 contadores, read y write, los cuales se encargan de indicar la posición a partir de la cual se debe leer o escribir en el mismo. Cuando el mismo se llena los contadores vuelven a cero, el uso es muy similar a un buffer circular.

Shell:

Nuestra Shell es la encargada de registrar y verificar si el usuario ingresa algún comando valido, realiza esta acción mediante la acumulación de caracteres leídos en un string el cual es comparado con nuestros comandos validos. Esta implementación deja abierta la posibilidad a futuro si alguien desea seguir agregando comandos validos solo deberá agregar un strcmp con su comando deseado y lo que realice el mismo. Para poder realizar esto fue necesario implementar funciones auxiliares tales como strcmp y strlen de la biblioteca string.

Abrir y cerrar disquetera

Las rutinas de enviado de paquetes mediante la interfaz ATAPI implicaron tomar precauciones debido a las diferentes arquitecturas Endian. A pesar que la gran mayoría de las fuentes en Internet sugerían tanto lecturas como escrituras en los puertos de la disquetera se hagan de a 2 bytes, decidimos realizarlo de a 1 byte a la vez. Al hacerlo de esta forma pensamos que nos evitariamos el posible conflicto entre arquitecturas Little y Big Endian. Nos encontramos con ciertos inconvenientes al intentar abrir y cerrar la disquetera en nuestra maquina real, cuando estas funcionalidades respondían correctamente en maquina virtual. Podemos mencionar que una de las posibles causas de esta anomalía era que la maquina virtual inicializa la memoria en cero y la maquina real no hace esto. Luego descubrimos que el problema se solucionaba haciendo lecturas y escrituras de a 2 bytes, tomamos como base que Intel trabaja con little Endian y la disquetera finalmente abrio y cerro en maquina real. Durante la implementacion se encontro con un problema durante la identificacion sobre cual de los diferentes puertos de ATAPI utilizar, debido a que el problema no puedo ser resuelto de manera correcta que funcionace para las pocas maquinas con procesadores de 32 bits (problema que fue resuelto para la ejecucion en maquinas virtuales) se utilizo la entrada esclava primaria, la cual era la que correspondia a la disquetera de todas las computadoras del laboratorio que compartian dicha arquitectura.

Funciones de test

Con el fin de que el usuario pueda probar algunas de las funcionalidades de nuestro código agregamos comandos que ayudan a visualizar su comportamiento. El comando ***testuno*** le da una bienvenida al usuario y le pide que tipee su nombre, mediante la combinación de funciones como scanf y printf, ambas de argumentos variables, mostramos su buen funcionamiento dándole una bienvenida personalizada.

El comando ***testdos*** realiza una secuencia de impresiones de flechas en pantalla, el fin de este comando es mostrar como los registros varían. Para ello, el usuario una vez ejecutado el comando, debe ir interrumpiendo esta acción mediante las teclas ctrl + r, sucesivas veces y así hacer visible el cambio en los registros.

testtres es el comando que imprime en pantalla 1500 veces una frase mientras realiza calculos cuyo resultado se ve reflejado en el registro ecx, si se lo interrumpe con la combinacion ctrl + r en cualquier momento de la ejecucion , es una variante interesante del comando ***testdos.***

Implementamos también el comando ***man***, el cual ayuda al usuario a identificar cuáles son algunos de los posibles comandos que están disponibles.

Los comandos ***clear***, ***clearup*** y ***cleardown***, son 3 comandos de comportamientos similar, el fin de ellos es limpiar la pantalla en distintos sectores.

Los comandos ***opencd, closecd*** e ***infocd*** son autodescriptivos en cuanto a su funcionamiento.

Bibliografia

“El lenguaje de Programacion en C” Brian Kernighan y Dennis Ritchie

http://wiki.osdev.org/ATAPI
http://lateblt.tripod.com/atapi.htm
http://suif.stanford.edu/~csapuntz/ide.html
https://github.com/qcho/arnix/blob/master/include/stdarg.h