Trabajo Práctico Especial

Arquitectura de Computadoras (72.08)

snmOS

Integrantes:

Sofía Carmen Picasso. [spicasso@itba.edu.ar](mailto:spicasso@itba.edu.ar). Legajo: 57700

José Martín Torreguitar. [jtorreguitar@itba.edu.ar](mailto:jtorreguitar@itba.edu.ar). Legajo: 57519

Ignacio Matías Vidaurreta. [ividaurreta@itba.edu.ar](mailto:ividaurreta@itba.edu.ar). Legajo: 57250

**Decisiones tomadas, Kernel:**

Modo video:

El mayor desafío enfrentado en el proyecto en cuanto al kernel concierne fue el de la conversión a modo video. En un principio se implementaron todas las funcionalidades en modo texto, sin embargo, ante la dificultad de la implementación del cambio a modo video en run time se optó por inicializar el sistema operativo en este modo. Esto se logra con el mero cambio de un bit en el archivo sysvar.asm.

Adaptar el sistema ya implementado para que funcionara en modo video requirió la comprensión del modo VBE (VESA bios extension). Debido a la implementación anterior del archivo naiveConsole.c, el cual era utilizado para escribir en modo texto previo al cambio, solo se requirió modificar la función “drawChar” para lograr el correcto funcionamiento del nuevo modo (lo cual no quiere decir que esto haya sido fácil). Se requirió el uso de un mapa que vincule caracteres con sus correspondientes representaciones en pixeles y el parseo de dicho mapa.

Esta modificación solo tiene contras en la dificultad de su implementación, puesto que en cuanto a los resultados estos resultan ser los de una UI más detallada y con mejor definición.

Resto del hardware:

Para el PIT se implementaron las funciones ya proveídas por la cátedra en el ejemplo del campus de la IDT y esto demostró ser suficiente para el buen funcionamiento del OS.

El teclado se implementó de tal manera que solo lee correctamente la disposición del teclado estadounidense estándar, aceptando tanto caracteres shift como non-shift. Los caracteres son guardados en un buffer longitud máxima 80 (al superarse esto el buffer comienza desde cero) de los cuales se lee mediante una system call, al hacerse esto el buffer elimina los caracteres leídos y mueve el buffer hacia atrás por la cantidad de caracteres que se hayan leído. Cualquier carácter que no tenga representación ascii o sea el shift izquierdo o derecho es leído, pero no utilizado, se decidió esto por si se decide agregarle funcionalidad a estos botones en el futuro.

IDT:

En la IDT se definen tan solo 5 posiciones del vector, aquellas correspondientes a las interrupciones de hardware del PIT y el teclado, las excepciones de división por cero y de código de operación inválido, y finalmente, la interrupción 0x80, la correspondiente a las system calls. Esto es lo único necesario para que el sistema funcione y se cumpla el enunciado pedido por la cátedra.

Excepciones:

Al ocurrir una excepción se indica de cual se trata (división por cero o código de operación inválido) y se imprimen los registros al momento de ocurrir la excepción. Puesto que al finalizar una excepción se retorna a la instrucción ejecutada antes de llamada esta se genera un ciclo sin final. Para solucionar esto se reemplaza en el stack pointer la dirección de retorno original por la función principal del kernel. Si bien esto soluciona el problema evita que en las funciones de usuario se pueda realizar alguna tarea que sea exclusiva del inicio como el texto introductorio que se muestra al iniciar el sistema operativo, por lo que esta funcionalidad fue movida al kernel en sí. Si bien esto viola la separación usuario-kernel, esto es un capricho menor de implementación del diseñador del kernel, quien quiso hacer el texto introductorio él mismo. Además esto quiere decir que no se puede retornar de la función main del kernel puesto que se ha perdido la dirección de retorno de esta función. Por esto nunca se retorna del Kernel si no que se llama a la función haltCPU que entra en un ciclo infinito el cual deshabilita las interrupciones.

System Calls:

Se definieron todas las system calls que se creyeron necesarias para el buen funcionamiento de las funciones usuario. Una lista completa de estas junto con las debidas aclaraciones puede ser encontrada en Kernel/Include/sysCalls.h. Se implementaron system Calls para la comunicación con todos los aspectos del hardware esto es: leer del buffer del teclado, escribir, o borrar la pantalla y hacer sonar o hacer dejar de sonar el speaker de la pc.

**Manual del usuario:**

Al abrir el kernel, aparecerá una pantalla de bienvenida. Luego de esta, se borrará la pantalla y aparecerá un prompt para empezar a utilizar el kernel. Una vez ahí, quizás el usuario no sepa que escribir, pero también está contemplado, pues en caso de que este este perdido al enviar un comando incorrecto se le recomendará utilizar el comando ‘help’.

Lista de comandos posibles:

* *help*: Imprime la lista de comandos que el usuario puede utilizar
* *exit*: sale del kernel.
* *echo*: Imprime en la terminal el mensaje escrito a continuación del echo. En caso de no haber escrito ningún mensaje, se decidió imitar el comportamiento de la terminal de Linux e imprimir una linea en blanco.
* *date:* Imprime la fecha y hora actual del sistema. IMPORTANTE: como se muestra al utilizar el comando, la hora que se imprime corresponde al huso horario de Greenwich, por lo que puede llegar a haber una diferencia con respecto al horario actual.
* *clock*: Abre una aplicación que muestra la hora en la terminal y se va actualizando. Apretando s cambia el color en el que se imprime la hora siguiendo el siguiente ciclo de colores: Blanco → Rojo → Verde → Azul → Blanco etc. Al cambiar el color genera un beep. Una vez en la aplicación de clock el usuario no puede escribir comandos hasta apretar q para salir de la aplicación
* *clear*: Limpia la pantalla
* *divByZero:* Genera una excepción del tipo división por cero. Agregado para demostrar que nuestro kernel soporta dicha excepción.
* *invOpcode*: Genera una excepción del tipo invalid opcode. Agregado para demostrar que nuestro kernel soporta dicha excepción.