|  |  |
| --- | --- |
|  | Al.Fa.Jor O.S. |
|  |  |

|  |
| --- |
| **[informe]** |
|  |

AROLFO, Franco

MOZZINO, Jorge

POMERANTZ, Alan

1. Decisiones e implementación del sistema
   1. Código

La codificación de este Trabajo Practico se realizo en C, en combinación de Assembler con nomenclatura Intel, combinando ambos lenguajes, y utilizando respectivamente cada uno para fines distintos, bajo nuestro criterio de cuál se adapta mejor a las necesidades que tuvimos a lo largo del desarrollo del programa.

La selección de dichos lenguajes fue propuesta por la Cátedra.

1.2. Compilación, linkedición y ejecución

Para poder compilar el Kernel Booteable, la cátedra otorgó un archivo de auto link edición y compilación denominado Arma y Compila respectivamente. Dichos archivos, cuyo contenido puede ser fácilmente editable, proveen las distintas opciones de linkedicion y compilación.

Utilizamos la opción -Wall para que muestre los Warning de compilación en C y Assembler, lo cual nos sirvió a la hora de programar y encontrar errores en una etapa temprana de desarrollo.

1.3. Pantalla

Hemos implementado en este apartado una pantalla de 80x25x2 bytes de tamaño , la cual representa el tamaño de una pantalla de resolución standard.

La misma implementa un cursor que acompaña, junto a las digitación del teclado, la posición donde se insertará el próximo carácter.

También se ha implementado el desplazamiento de la pantalla a medida que la misma se va llenando.

En función del parámetro Tickpos, al incluir el próximo caracter o línea, se realiza una comparación contra el tamaño de la pantalla, y se realiza o no, el correspondiente Scroll de la misma (teniendo en cuenta que se dejan las primeras dos filas para información del sistema).

Como medidas gráficas, y siguiendo la consigna, hemos implementado la inclusión de poder elegir, tanto BackGround Color, que modifica el color de fondo, y LetterColor, que modifica el color de las letras.

Available BackGround & Letter colors:

Black

Red

Green

Brown

Blue

Magenta

Cyan

White

1.4. Interrupciones

En el IDT (Interrupt Descriptor Table) ubicamos las entradas de las distintas interrupciones, numeradas bajo el setup\_IDT\_Entry manteniendo el orden visto en clase. Implementamos Timer Tick, teclado, INT80h (Entry Point del User Space al Kernell Space) y Exception de División por cero. Por decisión del grupo, no implementamos las demás entradas del Vector de interrupciones; se prevé en versiones futuras la inclusión para completar y otorgar más funcionalidad, aunque igualmente el programa funciona de manera correcta, sin que se manejen ciertos eventos que podríamos gobernar. No hemos movido las interrupciones para evitar el solapamiento con las excepciones, tampoco lo vimos necesario puesto a lo explicado anteriormente. También se prevé y considera en versiones futuras modificar dicho aspecto para perfeccionar aun más el desarrollo.

1.5. Shell

Desde el kernel se llama a la función shell(), con el que se puede comenzar el ingreso de comandos.

El mismo, entre otras cosas, inicializa el Buffer General (GeneralBuffer), llama a la INT80h para ubicar el cursor en la posición de inicio y opera con los comandos que se ingresen (simulando la espera con un ciclo infinito).

El shell cuenta con funciones para, en una primera instancia, obtener los comandos, para luego interpretarlos, ya sean validos o no, y realizar la funcionalidad asociada al mismo. Tener en cuenta que el máximo tamaño de un comando está delimitado por una constante simbólica nombrada BUFFER\_SIZE.

La llamada de comandos con más de un espacio, bajo nuestro criterio, fue considerado como un comando invalido. No obstante, la cantidad de argumentos para un comando no se reduce a la unidad, ofrecemos comandos que pueden llevar más de un argumento.

Ver el manual para obtener una descripción de la funcionalidad y utilización de todos los comandos disponibles en esta versión de Al.Fa.Jor. El intérprete de comandos informa al usuario si el comando es inválido o no. También provee un apartado de ayuda o help.

1.6. División del Kernel y user space. Interrupción INT 80h

Debido a las limitaciones con las que contamos a la hora de la división entre el Kernel y el User Space, hemos decidido delegar esa funcionalidad a la interrupción INT\_80h, similar a la de Linux, pero implementada por nosotros, la cual cuenta con diversas funciones.  
  
1.6.1 Funciones del Int 80h

Como selector de funciones utilizamos el registro extendido de 32bits EAX, que comienza en 0h (Por convención nuestra)

Entre las funcionalidad de la Int 80h podemos citar a las funciones \_\_write (int 80h 0), \_\_read (int 80h 1). Dichas funciones utilizan como parámetros los registros extendidos de 32 bits EBX, ECX y EDX. En estos registros debe estar un fd o file descriptor(dónde escribir), el Buffer y finalmente el tamaño de lo que se va a escribir. Utilizamos, por comodidad y estilo, un posicionamiento con un offset del EBP para referenciar a cada uno de los parámetros de las funciones, en vez de pushear en el stack los mismos.

Dichas funciones utilizan las funciones de la libreria estándar (implementadas por nosotros) para poder escribir y leer en pantalla.

Entre otras funcionalidades, citamos TickPos, la cual devuelve la posicion del próximo caracter a ser escrito, y PrintSomewhere, que dada una posición particular, imprime un arreglo de caracteres.

1.7. Printf(). Scanf(). PutChar() . GetChar()

Se implementaron las 4 funciones lo más similar al funcionamiento de la libreria standard.

Printf y Scanf se encuentran limitadas a la impresión únicamente de Enteros y Strings.

Se encadeno la llamada de printf y scanf con las funciones putchar y getchar, las cuales a su vez utilizan las funciones aun más básicas \_read y \_write.

1.8. Teclado

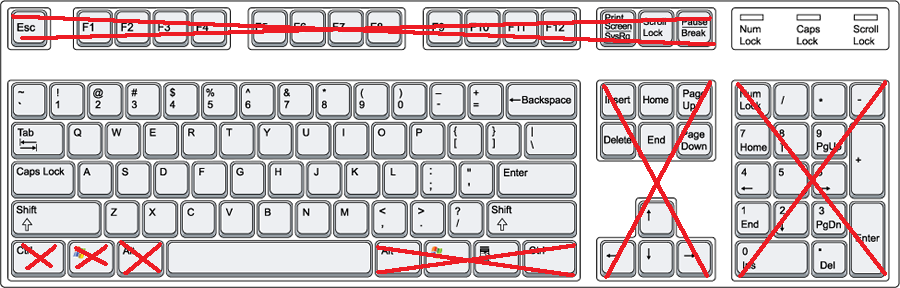
El teclado, en su implementación, cuenta con dos buffer. Uno denominado General, y otro de Entrada/Salida (E/S).

Las teclas al ser presionadas, ingresan al buffer general, cuya implementación es un buffer circular, para que luego el shell saque del mismo las teclas en el momento oportuno. El buffer de entrada y salida se utiliza para las funciones de dicha categoría. Ejemplo, getchar saca la primera tecla presionada del buffer de entrada y salida.

Asimismo, el teclado cuenta con dos diferentes idiomas, Español, el cual incluye acentos, eñe, y diferente mapeo de teclas. Y por lo contrario, se dispone del teclado Ingles, con otro mapeo de teclas diferente.

También se agregan las funcionalidades del Shift, para poner en mayúscula todos los caracteres, y agregar los caracteres especiales del teclado, sea signos de exclamación, pesos, etc. Además, se han implementado BloqMayus y Tab para agregar tabulaciones, entre otros comandos propios y comunes de un teclado standard.

En el siguiente dibujo se tachan las teclas que no tienen relevancia en este trabajo practico y las cuales no garantizamos funcionamiento esperado alguno.



1.9 Morse code vía Speaker.

En base a lo solicitado por la consigna, se incluyó la posibilidad de representación sonora de los caracteres.

Esto se realiza mediante la selección del modo MorseCode, con el comando de la shell denominado 'morse'.

Toda cadena de caracteres (solo letras) a la derecha del comando, se convierte en su correspondiente equivalente en Morse.

En base a un gran número de experimentaciones en la selección de una frecuencia audible adecuada que permita la correcta interpretación del código, finalmente concluimos que: los 440hz (aorrespondiente a una nota musical La Central), con un correspondiente tamaño de pausa entre caracter y caracter, y que triplicar el tamaño de una Raya en morse contra un punto, eran los indicados para tal fin.

Se han agregado funcionalidades especiales para el manejo del Speaker, como seleccionar un código Morse visual, mediante el comando visualmorse, el cual imprime el código Morse al mismo tiempo que lo amplifica por la salida.

También se puede seleccionar la velocidad de pitido, manteniendo por default la seleccionada bajo nuestro criterio.

1.10 Timer Tick.

Hemos adoptado la decisión que el TimerTick gobierne ciertos aspectos del funcionamiento del Kernel.

Gestiona la función wait(), que permite simular una pausa estipulada, y también maneja al RTC (Real Time Clock) actualizándolo segundo a segundo. La frecuencia del TimerTick es independiente del hardware y está fija en 55ms, lo cual independiza la temporalidad de la ejecución de las funcionalidades, sin tener en cuenta en que hardware se desarrolle.

1.11 Manejo de memoria.

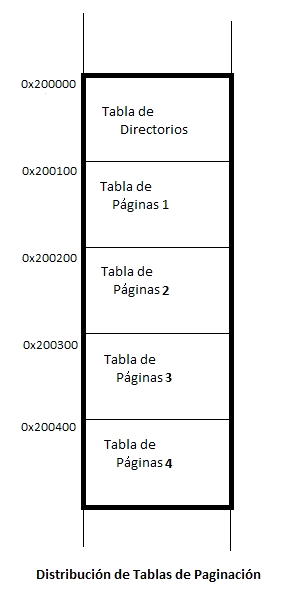
El grub brindado por la cátedra nos deja el kernel para su uso en modo protegido y Flat, es decir, un solo segmento. Al setear en 1 el bit 31 del registro CR0 del micro, éste activa el modo de paginación, teniendo el registro CR3 la dirección de la tabla de Directorios.

A la hora de distribuir la memoria para el uso del usuario y del sistema propusimos la siguiente solución:

* Paginar (dividir la memoria en bloques de 4KB) hasta 16 MB.
* Tomar 4MB para el uso del sistema, y los 12MB restante para el usuario.
* Colocar la tabla de Directorios y las 4 páginas correspondientes al diseño propuesto en paginas contiguas, a partir de la dirección 0x200000.

Los 4MB para el uso del sistema son más que suficientes en nuestro trabajo practico, y dejamos memoria reservada para el sistema para un uso futuro.

El diagrama de memoria quedaría entonces de la siguiente manera, representando los 4KB que contienen la tabla de Directorios y las 4 páginas contiguas que paginan los 16MB del sistema:



Llegado el momento de pedir un espacio de memoria para con el malloc, decidimos dar de a paginas, es decir, si me pedían una cantidad de memoria que no era múltiplo de 4KB, devolvíamos la cantidad de memoria pedida mas la necesaria para llegar a completar la página completa.

Nótese que con esta implementación tenemos la desventaja de perder memoria, pues si me pidiesen 1Byte, estaría dando 4KB.

Por lo tanto, como estoy devolviendo tantas páginas de memoria como me pidieron, decidimos utilizar los bits AVAIL de las entradas de las tablas de Páginas, con la siguiente convención:

* AVAIL = 001b --> MEMORIA OCUPADA
* AVAIL = 000b --> MEMORIA LIBRE

Ahora se nos presentaba el siguiente inconveniente: si hacían dos mallocs seguidos y hacían free de uno de ellos, no tengo manera de saber cuántas páginas de memoria reserve para esa dirección.

Nuestra solución fue la siguiente: una estructura auxiliar (Array) en la cual guardaba las direcciones de memoria reservadas y la cantidad de páginas que tenía cada una.

Decidimos también, en esta implementación, dejar el tamaño de dicha estructura fijo, en particular, de 1024 posiciones. Así, nuestro sistema soporta solo 1024 direcciones de memoria reservadas a la vez.

1.12 CD Booteable.

Para la edición y quemado del CD Booteable con el Kernel AlFaJor, hemos utilizado la herramienta de .iso: mkisofs, con la siguiente secuencia de comandos

-pad -b tpe.img -R -o tpe.iso ./tpe.img

Lo cual nos produjo un archivo tpe.iso, el que finalmente terminamos grabando en un CD-RW Verbatim 700MB 8-12x, con la utilidad Brasero, para obtener el CD finalizado y correctamente funcionando.

2. Referencias y bibliografía consultada

Incluimos referencias en general a páginas de consulta para poder evadir los distintos problemas de desarrollo que se nos presentaron a la hora de desarrollo. Más que nada en la implementación del Speaker

Hemos encontrado muy valiosa las páginas web

http://www.daniweb.com/

http://www.osdev.com/

http://www.wikipedia.com/

Donde hemos encontrado cosas para el speaker <http://www.daniweb.com/software-development/c/code/216802> y algunas cosas de Libreria standar, como implementaciones alternativas de putchar, scanf, tolower, atoi, scanint, etc.

También hemos obtenido la tabla de conversión morse de una implementación de dicho programa en la página ya mencionada.

Además, hemos consultado con los ayudantes de cátedra, y otros grupos de desarrollo del TPE, los cuales nos dieron una mano a la hora de ciertos inconvenientes con las frecuencias audibles del speaker, problemas de conversión, pasaje de parámetros y casteos, y discutimos mejoras en la implementación, performance, sugerencias de desarrollo, herramientas de debugging, etc.