Arquitectura de Computadoras

Trabajo Práctico Especial

Informe

Instituto Tecnológico de Buenos Aires

Grupo 22

Integrantes:

- Barnatán, Martín Alejandro (64463)
- Garrós, Celestino (64375)
- Pedemonte Berthoud, Ignacio (64908)
- Weitz, Leo (64365)

Fecha de Entrega: 4 de junio de 2024

2024

Índice:

Objetivo

El objetivo de este proyecto fue implementar una parte de un Kernel, el cual es booteable mediante el software-loader “Pure64”, para que administre los recursos de Hardware de una computadora. Además se le debe brindar a los potenciales usuarios una API para que sea posible hacer uso de dichos recursos desde el User Space.

Para lograr esto, se debió separar claramente la memoria en una sección destinada para el kernel (Kernel Space) y otra para futuros usuarios (User Space). La primera interactúa directamente con el hardware mediante el uso de drivers que faciliten la comunicación con los periféricos (teclado, pantalla, etc.). Por otro lado, la segunda se comunica con estos utilizando las *system calls* brindadas por la primera. Se creó una librería, la cual intenta asemejarse a la librería estándar de C en Linux y que implementa las mismas funciones que esta.

Esta implementación cuenta también con un intérprete de comandos similar a una Shell que contiene una serie de funcionalidades. Entre estas se encuentran:

- Función de ayuda
- Funciones que permitan testear la excepción de división por cero y de código de operación inválido
- Comando para desplegar la hora del sistema
- Comando para obtener los valores de los registros en cualquier momento que se desee
- Comando que permita jugar al juego “Eliminator”
- Comando que permita agrandar o reducir el tamaño del texto de la pantalla
- Comando de música “Nano Song”

Separación User Space - Kernel Space

La finalidad del kernel es administrar correctamente los recursos de la computadora, así como brindarle al usuario herramientas para que pueda hacer uso de estos. Es decir, el kernel monopoliza el acceso al hardware y crea una API para el usuario, evitando que este tenga acceso directo al hardware.

Esta API consiste en la implementación de diversas System Calls, las cuales se encuentran programadas y almacenadas dentro del Kernel Space, más específicamente en la IDT. Estas son accesibles por el usuario mediante la instrucción “INT 80h” y reciben una cantidad de parámetros variable, entre los que se encuentra el “File Descriptor”, que indica a qué System Call se quiere llamar.

Con dicha base, se crearon diversas librerías que facilitan aún más el uso del hardware para el usuario. Entre estas, hay una que intenta asemejarse a la estándar de C en linux y cuenta, por ejemplo, con las funciones: printf, scanf, putChar y getc, entre otras.

System Calls

Para las System Calls, lo primero que se hizo fue cargar la interrupción en la IDT en la entrada 80h, apuntando a la rutina de atención de interrupción encargada de ejecutar la System Call correspondiente. Se desarrollaron las System Calls expuestas en la siguiente tabla:

id	nombre	rbx	rcx	rdx
3	sys_read	FileDescriptor fd	char * buf	uint64_t count
4	sys_write	FileDescriptor fd	const char * buf	uint64_t count
5	sys_draw_rectangle	Rectangle *r	-	-
6	sys_rtc	Timestamp *ts	-	-
7	sys_sleep	uint32_t ticks	-	-
8	sys_clear	-	-	-
9	sys_play_sound	uint64_t ticks	uint64_t hz	-
10	sys_change_font	-	-	-
11	sys_registers	uint64_t *arr	-	-
33	sys_nano_face	-	-	-

Las system calls descritas en la tabla anterior realizan las siguientes operaciones:

- *sys_read:* recibe un FileDescriptor *fd*, un buffer *buff* y una cantidad de caracteres a leer *count*. Guarda en *buf* a lo sumo *count* caracteres que estén guardados en el buffer. Retorna la cantidad de caracteres leídos.
- *sys_write:* recibe un FileDescriptor, un buffer y una cantidad de caracteres a escribir. Escribe en pantalla en base al FileDescriptor indicado a lo sumo *count* caracteres de *buf*. Retorna la cantidad de caracteres escritos.
- sys_draw_rectangle: recibe un puntero a Rectangle *r*. Dibuja en pantalla un rectángulo con la información brindada por el contenido apuntado por *r*.
- *sys_rtc:* recibe un puntero a Timestamp *ts*. Guarda en la dirección apuntada por *ts* la información correspondiente a la fecha y hora del sistema.
- *sys_sleep:* recibe un número entero positivo *ticks*. Espera que transcurra *ticks* unidades de tiempo.

- *sys_clear:* limpia el contenido de la pantalla.
- *sys_play_sound:* recibe un número entero positivo *ticks* y un número entero positivo *hz*. Emite sonido durante *ticks* unidades de tiempo a una frecuencia *hz*.
- *sys_change_font:* cambia el tamaño de fuente del driver de video.
- *sys_registers:* recibe un array *arr*. Guarda en *arr* el estado de los registros.
- *sys_nano_face:* Imprime en el display una foto de Fernando Alonso.

En la tabla, FileDescriptor es un enum cuyos valores posibles son *STDOUT* (salida estándar, valor 0), *STDERR* (salida de error, valor 1) y *STDMARK* (marca de pantalla, valor 2). Rectangle es una estructura que describe un rectángulo en base a su posición superior izquierda, su posición inferior derecha y su color. Finalmente, Timestamp es una estructura que describe una fecha y hora.

Manejo de Interrupciones

Para el manejo de interrupciones, se contemplaron el *Timer Tick* y la *Keyboard Interrupt*. Las mismas fueron cargadas en las entradas 20h y 21h de la *IDT* respectivamente, con sus rutinas de atención de interrupción. La máscara que se estableció para el *PIC* es FCh, de manera que sólo las dos interrupciones mencionadas estuvieran habilitadas.

Se cambió la frecuencia del *Timer Tick* a 120Hz, con el objetivo de optimizar el módulo eliminator y se desarrolló el método sleep, cuyo objetivo es esperar una cierta cantidad de *ticks* de unidades de tiempo.

En cuanto a la interrupción de teclado, se desarrolló el *Keyboard Driver*, cuyo objetivo es capturar las teclas que son presionadas. Las funcionalidades desarrolladas en el mismo son utilizadas activamente por los distintos módulos del sistema.

Drivers

Driver de Video

Para comenzar con la implementación del driver de video se necesitó encontrar una manera de imprimir en pantalla caracteres, emulando la funcionalidad de la naive console presente en el proyecto BareBones.

Se partió del driver de video proporcionado por la cátedra y se utilizó el programa bdf2c (https://github.com/pixelmatix/bdf2c) para realizar la conversión de la fuente Spleen (https://github.com/fcambus/spleen) en sus tamaños de 8x16 y 16x32 a arreglos de caracteres procesables de manera sencilla desde el código del driver.

Por defecto al convertir la fuente con la aplicación se genera una lista de *defines* que permite que la fuente sea legible y fácilmente modificable desde un editor de texto. Cada char en este arreglo representa bit a bit la presencia o ausencia de un píxel en una línea de 8 píxeles de ancho. Recorriendo estos arreglos (y haciendo los ajustes que correspondan al usar la fuente de 16 x 32 píxeles) se puede dibujar pixel a pixel cada caracter *ASCII* necesario para mostrar texto. Se guarda además una posición en x y en y del cursor para poder realizar la impresión normalmente, además de casos especiales para caracteres como el newline.

También se implementó una funcionalidad de scroll en el driver de video, donde al llegar al final de la pantalla se mueven los datos del buffer de video hacia atrás, efectivamente moviendo los contenidos de la pantalla una línea hacia arriba.

Por último, para facilitar el dibujado de la pantalla libre se implementó una función *drawRectangle* que permite dibujar un rectángulo dado su color y sus coordenadas. Esto se utilizó posteriormente para la syscall *sys_draw_rectangle*, que utiliza el eliminator.

Driver de Teclado

Para el manejo de las interrupciones de teclado, se decidió implementar una matriz *keyValues* tomando las filas como los valores de las teclas y dos columnas, una con la representación directa y otra en caso de que el *shift* o *bloq. mayús.* estuviera accionado.

Los valores recibidos de la interrupción se ingresan en un buffer circular manejado con el método FIFO (First In, First Out), es decir que el primer carácter que ingresa al buffer será el primero en salir del mismo. Para esto, primero se analiza si se trata de algún caracter especial como ser una tecla *shift* o *alt*, en cuyo caso se realiza la operación correspondiente, como es posible observar en el código. En el caso de haber sido apretada la tecla *esc*, se llama a un método para guardar los registros, pues se definió que la misma sea la encargada de permitirle al usuario realizar dicha operación.

Como al soltar una tecla también se genera una interrupción, sería deseable que dichos valores no sean escritos en el buffer. Por lo tanto, se compara con 0x70 que es el máximo valor de tecla apretada como se pudo observar en la documentación. Luego, se chequea que no sea un caracter especial ni una tecla de función rápida (*F1*, *F2*, etc.) pues se definió que estas no tuvieran ninguna función en el Kernel. Por último, se hacen validaciones para saber si se debe colocar en el buffer el caracter o su equivalente con *shift* presionado.

Finalmente, se implementó una función para obtener el siguiente caracter del buffer, para poder realizar la lectura de los valores guardados por la función anterior.

Driver de Sonido

El altavoz de una computadora tiene 2 estados posibles: in y out. Si este pasa de estar “in” a “out” y viceversa con una cierta frecuencia, entonces la tarjeta de sonido emite un sonido en la misma. Por lo tanto, para implementar un driver de sonido simplemente debimos manipular dicho estados para que cambiaran a una frecuencia tal que emitiera una vibración audible.

Esto se realizó teniendo en cuenta que el output 2 del PIT (Programmable Timer Interval) se puede conectar al puerto del altavoz (61h) para que modifique su estado en cada *tick*. Al hacer esto (lo cual se logra seteando el bit 1 del puerto 61h a 0), lo único que queda por hacer es setear de 0 la velocidad a la que el output 2 del *PIT* “tickea”. Esto se puede hacer a través de los puertos 42h y 43h.

Sin embargo, también hay que tener en cuenta que el altavoz de la PC tarda aproximadamente 60 millonésimas de segundo en cambiar de estado. Por lo tanto, si este último se modifica, y se vuelve a cambiar en menos de 60 microsegundos, el altavoz no tiene el suficiente tiempo para verdaderamente cambiar su estado. Además, ya que estamos trabajando con QEMU en vez de con hardware real, debemos correrlo con el comando “-audiodev pa,id=speaker -machine pcspk-audiodev=speaker” para simular la tarjeta de sonido.

Por lo tanto, se implementó una función que cambiara la frecuencia del output 2 del PIT y vinculara el altavoz con dicho output, junto con funciones que los desvincularan y que esperaran (para darle un tiempo al sonido). Para esto nos basamos en el código de muestra mostrado en osDev (https://wiki.osdev.org/PC_Speaker#How_It_Works).

Excepciones

En primer lugar, se desarrollaron las rutinas de atención correspondientes para ambas excepciones (dividir por cero y opcode inválido). En ambos casos se imprime un mensaje de error indicando por qué se produjo, y luego se reutilizó la funcionalidad de salvar el estado de los registros y mostrarlos en la pantalla.

Para esta última funcionalidad, se debió copiar el print del *registers* en *Userland* pero tomando las funciones de imprimir del kernel, dado que sino no se podía realizar esta operación. Si se salvaban sin imprimir, el usuario podría apretar *esc* pisando dicho backup, y luego no los podría obtener. Por esto se decidió implementar una función print igual en el archivo de excepciones.

Por último se creó la función para retornar al contexto del usuario una vez impresos los registros. Además, se desarrolló un *exceptionDispatcher* para ser llamado desde Assembler y ejecutar la rutina correspondiente. Allí, se replicó la macro *Handler* de las interrupciones pero para las excepciones. Con esto, se cargó finalmente ambas excepciones en la *IDT*.

Shell

El intérprete de comandos se implementó con un loop infinito que se basa principalmente en diversas llamadas a funciones de la librería estándar. En primer lugar se lee de ella una línea ingresada por el usuario (hasta la aparición de un “*n*” en el buffer de teclado). Dicha lectura se realiza caracter a caracter, aprovechando que de esta manera se puede a la vez imprimir cada lectura.

Luego, la línea es interpretada comparándola con cada uno de los comandos aceptados por la Shell. De esta comparación se obtiene un número identificatorio del comando a ejecutar, el cual se utiliza dentro de un *switch* para determinar la función que se debe llamar. En caso de que el string ingresado no se corresponda con ningún comando ofrecido, simplemente se le informa de esto al usuario.

Las funciones que representan cada comando pertenecen en su gran mayoría a la librería estándar, ya que necesitan interactuar con el hardware del sistema, ya sea para obtener ciertos datos, como los valores de los registros, o para acceder a periféricos como el altavoz.

Juego - Eliminator

Para el juego de eliminator se planteó una matriz de estado del tamaño del entorno de juego, que guardaría en cada posición si la misma estaba o no ocupada. Se procedió a setear un tamaño de pixel, creando una función para dibujar cuadrados de tamaño fijo emulando una resolución más baja.

Posteriormente se implementó el menú de opciones, el sistema de controles y el dibujado de juego con las syscalls creadas. Se mantiene la posibilidad de cambiar el tamaño del campo de juego y de píxel con los defines del inicio del programa.