Instituto Tecnológico de Buenos Aires

72.08 - Arquitectura de Computadoras

TPE - INFORME

Integrantes:

Santiago Rivas - 61007 - srivas@itba.edu.ar

Franco Panighini - 61258 - fpanighini@itba.edu.ar

Jose Leon Burgos Sosa - 61525 - jburgos@itba.edu.ar

Entrega: 01/11/2022

Índice

Introducción

El objetivo del TPE es la implementación de un kernel booteable por *Pure 64*, que administre los recursos de hardware de una computadora y muestre características del modo protegido de Intel. Debe proveer una API para que aplicaciones de usuarios puedan utilizar estos recursos. Para ello se definen dos espacios separados, el *kernel space* y el *user space*. En el *kernel space* se implementan drivers, mediante los cuales se interactúa directamente con el hardware, y además, estos drivers proveerán las funciones que utilizará el usuario desde el *user space*.

Entorno de Trabajo

El desarrollo del TPE se llevó a cabo utilizando los editores NeoVim y VSCode, para el control de versiones se utilizó Git con un repositorio de GitHub.

Para compilar y correr el TPE se siguieron las instrucciones provistas por la cátedra. Se utilizó *docker* con una imagen provista y *qemu* para correrlo.

Diseño

Separación del Kernel - Userspace

En el *kernel* se almacenan y administran los recursos de la computadora, estos no deben ser utilizados ni modificados por usuarios sin permisos. Para acceder a los recursos disponibles se utilizan interrupciones de software, *int 80h*, o de hardware, pulsar una tecla. En el *user space* se hace uso de los recursos de la computadora utilizando interrupciones, las cuales invocan system calls, que permiten al usuario desarrollar funciones más complejas. Entonces la separación consiste en un kernel independiente al user space, y un user space que depende de las interrupciones y system calls provistas por el kernel.

Interrupciones

El primer paso en el manejo de interrupciones es la carga de la Interrupt Descriptor Table con las distintas interrupciones y excepciones que se utilizaran. Las entradas se insertan en idtLoader.c y las rutinas de interrupción se definen en interrupts.asm.

El manejo de interrupciones, se administra mediante las funciones definidas en *interrupts.asm* dentro de *kernel space.* El correcto funcionamiento depende de que esté correctamente cargada la IDT. En este caso se manejan tres interrupciones:

- ***_irq00Handler*** *-> timer tick*, el cual maneja la cantidad de ticks que se realizan desde que se enciende el procesador. Un caso de uso es el reloj.
- ***_irq01Handler*** *-> keyboard*, maneja las interrupciones de teclado. Un caso de uso en este TPE es el control de la dirección de los jugadores en el juego Tron.
- ***int 0x80*** *-> sys_calls*, manejan interrupciones específicas como lectura, escritura, entre otras. Estas interrupciones se pueden encontrar en funciones dentro de *lib.c* en *userland*, como ser getChar o printf.

Excepciones

El manejo de excepciones es muy similar al de las interrupciones. Se cargan en la IDT y luego se administran y ejecutan por la función *exceptionDispatcher*, definida en *exceptions.c*. Se manejan dos excepciones:

- divideZeroException.
- invalidOpCodeException.

Ambas excepciones muestran mensajes de error por pantalla e información de los registros incluido el instruction pointer en el momento de la excepción. Posteriormente se muestra un contador y se regresa a la Shell.

Syscalls y drivers

Se categorizaron las system calls según su funcionalidad en distintos drivers. Al ejecutarse una syscall, se lanza una interrupción mediante la cual el kernel interpreta que recurso se quiere utilizar, y así invocar a la función adecuada del driver pertinente.

El syscallManager (*kernel*) se encarga de solicitar los recursos necesarios al realizar una interrupción, invocando a la función del driver que corresponda.

Los drivers son bibliotecas que contienen todas las funciones pertinentes a una funcionalidad particular del hardware. Los drivers disponibles dan acceso al uso y modificación de los píxeles de la pantalla, la utilización del teclado, el sonido, y al manejo y acceso del tiempo.

Timer

Este driver se encarga del manejo y acceso a funciones relacionadas con el tiempo.

Consiste de las funciones:

- *timer_handler*: actualiza la cantidad de ticks que han pasado.
- *ticks_elapsed*: retorna la cantidad de ticks ocurridos hasta ese momento.
- *seconds_elapsed*: retorna la cantidad de segundos ocurridos hasta ese momento calculados a partir de la cantidad de ticks (sabiendo la duración de un tick).
- *wait*: se le pasa un entero por parámetro que representa la cantidad de tiempo, que se debe esperar.
- *milliseconds_elapsed*: retorna la cantidad de milisegundos ocurridos hasta ese momento calculados a partir de la cantidad de ticks (sabiendo la duración de un tick).
- *getTime*: retorna un valor de tipo *long* el cual contiene lo retornado por las funciones getHours, getMinutes y getSeconds definidas en assembler en *libasm.asm.*
- *getDate*: retorna un valor de tipo *long* el cual contiene lo retornado por las funciones getDay, getMonth y getYear definidas en assembler en *libasm.asm.*

KeyboardDriver

Este driver se encarga de la administración y utilización de los recursos de teclado. Se implementó en Assembler la función *keyPressed*, que se encarga de activar el teclado y obtener el scan code de la tecla presionada.

Al presionar una tecla se genera una interrupción que ejecuta una función de C llamada ***_irq01Handler***, que recibe el scan code de la tecla, lo traduce a un código ASCII y lo guarda en una estructura de tipo buffer.

Una vez que se obtiene la tecla presionada en ASCII, se guarda en un buffer que tiene una estructura FIFO para que el usuario pueda acceder al carácter a través de la función *getChar*, utilizada como syscall. En caso de que se utilice *getChar* y el buffer se encuentre vacío, se retorna un 0 para que no se realice una espera activa. Esta decisión de diseño fue tomada para que al ejecutarse esta función de userland no se deba esperar a que se ingrese una tecla, sino que pueda seguir ejecutándose código.

Consiste de las funciones:

- *saveKey*: se le pasa por parámetro un scan code, el cual verifica que sea válido, y utilizando la función getKey guarda el valor ascii de la tecla presionada en el buffer.
- *readBuf*: se le pasa por parámetros un puntero a la primera posición de un string, en el cual se debe cargar el contenido de buffer, y un entero que determina la cantidad de caracteres que se deben cargar del buffer al string. Retornando la cantidad de caracteres que se pudieron cargar.
- *clearKeyboardBuffer*: vacía el buffer.
- *getCount*: retorna la cantidad de elementos actualmente guardados en buffer.
- *getKey*: se le pasa por parámetro el scan code de la tecla y esta función la asocia a su código ascii y lo retorna.

Video Driver

Driver encargado de administrar la manipulación de la pantalla, utilizando *VESA* activado desde *bootloader*. El mismo se encarga de manipular los píxeles visibles, los cuales pesan tres bytes, debido a la composición de colores BGR (*blue*, *green*, *red*).

Consiste de las funciones:

- *fillRect*: función encargada de rellenar un rectángulo con un color de un tamaño determinado por parámetros.
- *printSquare*: rellena un cuadrado con un color determinado.
- *printChar*: mediante una *true font* definida en *font.h* se calcula cual es el carácter que se debe imprimir, para la impresión de dicho carácter se llama a la función *printSquare*.
- *printString*: wrapper escribe un *string* comenzando en la coordenada (0, 0).
- *printStringAtX*: wrapper permite escribir un *string* en una posición desfasada en la coordenada “x”.
- *printStringAtX:* llama a *printChar* hasta encontrar la terminación del string (*‘\0’*) y maneja el espacio disponible en pantalla.
- *getPixel*: retorna el color del píxel correspondiente a una coordenada recibida por parámetro.
- *colorScreen*: función encargada de definir el color del fondo.
- *clearScreen*: función que emula el comando clear de Linux.
- *getHeight*: retorna la altura de la pantalla.
- *getWidth*: retorna el ancho de la pantalla.
- *changeFontSize*: permite cambiar el tamaño del font.

Para lograr el cambio de tamaño en la fuente se multiplicó por un valor por default uno el ancho y alto del carácter definido en *font.h*.

Syscalls

Las variables c de tipo *Color* es una estructura creada para almacenar el color formado por BGR.

Número	Syscall	Descripción
0x00	long sys_write(unsigned char fd, char *s, Color c)	Imprime el parámetro *s* en la salida indicada por *fd*. Si es salida estándar imprime el color indicado, si es salida por error, el mensaje se imprime en rojo
0x01	long sys_read(unsigned char fd, char *s, int count)	Lee una cantidad *count* por la entrada indicada por *fd, y* lo guarda *en la dirección de memoria s*.
0x02	long sys_writeAt(short x, short y, char *s, Color c)	Imprime el string con un color en una posición “x” e “y” determinada.
0x03	long sys_clearScreen()	Borra todo lo pintado en pantalla.
0x04	long sys_wait(int millis)	Espera un tiempo millis, deteniendo al procesador con *hlt.*
0x05	long sys_time()	Retorna el tiempo.
0x06	long sys_date()	Retorna la fecha.
0x07	long sys_getScreenHeight()	Retorna el alto máximo de la pantalla.
0x08	long sys_getScreenWidth()	Retorna el ancho máximo de la pantalla.
0x09	long sys_timeRead(unsigned char fd, char *s, int count, int millis)	Similar a sys_read, pero espera un tiempo *millis* a recibir una entrada. En caso de recibir un enter, se corta la lectura. Retorna la cantidad de caracteres leídos.
0x0A	long sys_drawRectangle(int x, int y, int width, int height, Color c)	Dibuja en pantalla un rectángulo de color *c* en la posición (x, y) con un ancho *width* y una altura *height*.
0x0B	long sys_changeFontSize(int diff)	Cambia el factor de escala de la fuente del driver de video. Si el valor recibido es positivo se incrementa al factor escala, caso contrario se decrementa.
0x0C	long sys_inforeg(long *registers)	Llena un arreglo de 17 posiciones con los valores de los registros.
0x0D	long sys_beep()	Activa el beeper de la computadora con una frecuencia constante.

Shell

Es un interpretador de comandos que corre continuamente luego de ser llamado por el kernel. Puede interpretar diferentes comandos para ejecutar programas, sirve como interfaz entre el usuario y los programas disponibles.

La shell lee del buffer de texto muchos caracteres a medida que se van escribiendo. Además guarda en un buffer (y borra en el caso de backspace) los caracteres ingresados. Este buffer luego es analizado utilizando la función de la biblioteca estándar strcmp, y se compara con los nombres de los diferentes comandos. Cuando un comando introducido no es válido o un parámetro no está en el formato correcto se indica por salida de error. Toda la impresión en pantalla es manejada por el driver de vídeo.

Comandos:

1. Help:
   Lista los comandos disponibles junto a una breve descripción de cada uno.

2. Tron:
   Llamado al juego implementado tron. También reduce el tamaño de la fuente para que las palabras impresas por el juego entren correctamente en la pantalla. Luego, retorna a la shell dejando el tamaño de la fuente como estaba.

3. Inc-font:
   Incrementa el tamaño de la fuente. En el caso de no ser posible (determinado por el driver de video) envía un mensaje de error por salida estándar.

4. Dec-font:
   Decrementa el tamaño de la fuente. En el caso de no ser posible (determinado por el driver de video) envía un mensaje de error por salida estándar.

5. Inforeg:
   Imprime en pantalla los valores guardados por los registros en un momento dado. Este momento se puede indicar presionando y liberando la tecla control. Si esto no ocurre antes de llamar a inforeg entonces los registros aparecerán todos con cero.

6. Printmem:
   Imprime lo contenido dentro de los próximos 32 bytes de memoria recibido por parámetro. El parámetro enviado desde la terminal debe ser un número hexadecimal y se indica prefijando con "0x" y hay un límite superior en las direcciones posibles.

7. Time y date:
   El comando time imprime el tiempo actual en el formato hh:mm:ss. El comando date imprime la fecha actual en el formato de/mm/aa. El comando datetime imprime la fecha y la hora en el formato dd/mm/aa hh:mm:ss

8. Clear:
   Borra todo lo impreso en pantalla y deja el cursor en la posición más alta a la izquierda.

9. Exit:
   Cierra la shell y se va de userspace retornando a kernel space y finalizando el programa.

10. Div-zero e invalid-op:
    Comandos que llaman a códigos que causan las respectivas excepciones. El kernel space maneja la excepción y luego de un tiempo una nueva shell es generada para que el usuario pueda seguir utilizándose.

Tron

Para la implementación del juego “Tron Light Cycles” se empleó el uso de una estructura *Canvas* que donde se almacenan las dimensiones del *game board* y una matriz del mismo. Se escaló la matriz o *game board*, lógico y los puntos que ocupaban los jugadores de manera tal que fueran equivalentes al dibujarse en pantalla. Durante el juego primero se actualiza la dirección a la que apunta cada jugador, luego se verifica que la nueva posición se valida. Se verifica que no se haya chocado con el perímetro, ni contra el otro jugador. Después, si ningún jugador perdió, se actualiza el Canvas. Se implementaron funcionalidades útiles como: pausar el juego durante la partida, reiniciar al final o durante una partida, un contador de la cantidad de partidas que ganó cada jugador, y la opción de resetear a cero. Todas estas funcionalidades están debidamente informadas, en los momentos pertinentes, a los jugadores a través de rótulos. Desde el punto de vista gráfico se eligieron los colores clásicos del juego y se añadió al inicio un contador y un cartel titilante.

Limitaciones:

La shell tiene un límite en la escritura de comandos, ya que los comandos permitidos son cortos y pocos.

Otra limitación es el pasaje de parámetros. Como solo un comando recibe parámetros no se implementó un método general para la recepción de parámetros. El comando printmem recibe parámetros, pero estos no son validados por la shell. Esta es la limitación más grande de la shell ya que la hace poco extensible. Esta funcionalidad no se desarrolló ya que no fue necesitada extensamente.