Informe Reentrega TPE

Autores:

- Huerta Joaquin (64252)
- Kaneko Tomas (62297)
- Valentinuzzi Antonio (64751)

Grupo: 5

Fecha de entrega: 24 de noviembre de 2024

Arquitectura de Computadoras - 2C 2024

Índice

Resumen

El objetivo de este trabajo práctico fue el de crear un sistema basado en PURE64, aplicando los conocimientos adquiridos durante la cursada de la materia.

El sistema arranca desde Kernel space donde luego se va a User space para la ejecución del shell, el cual es la principal herramienta de comunicación entre el sistema y el usuario. Esta shell, junto con otro programa que se encuentra en User space, snakeApp, emplean una librería, GlobalLib, la cual es la que tiene la mayoría de funciones que se pueden correr desde el User Space. A partir de esta se pueden hacer llamadas a system calls.

Para la implementación del sistema, también se implementaron excepciones de tipo 0 y tipo 6 en caso de que estas sean cometidas, a su vez que interrupciones de system calls y de drivers. Estas se encuentran en Kernel Space.

Explicación

Durante el desarrollo, primeramente se trató de buscar el correcto funcionamiento de las funciones que tendrían que utilizar “Drivers”, por lo que hubo cierto desarrollo de una librería paralela, NativeLib, exclusiva de Kernel space.

Drivers de Video:

Con respecto al driver de video, se decidió la implementación de ‘bitmap fonts‘ para la impresión de caracteres por pantalla, debido a que se considero lo más sencillo. Esto a su vez causó que, para la impresión de caracteres más grandes (16x16) o chicos (8x8) se tuvieran que buscar distintos bitmaps de caracteres. La razón por la que se decidió de tener 3 tamaños fue para que a la hora de iniciar en una letra de tamaño mediano (12x8) se tenga la posibilidad tanto de agrandar como de achicar el tamaño de letra. Sin embargo, a pesar de que es más sencillo, ocupa varios archivos solamente llenos con los mapas, los cuales si no fueran por los comentarios, serían muy poco claros.

Por otro lado, para el dibujo de figuras geométricas, se implementó el dibujo de rectángulos, puesto a que estos incluyen también cuadrados. El dibujo de círculos fue una implementación tardía que surgió solamente a la hora de desarrollo del snake para dibujar la fruta, la cual originalmente era un cuadrado rojo. Esto último, a pesar de ser relativamente sencillo, es muy limitante, por lo que si se continuara el desarrollo del sistema se debería reemplazar por una que pueda dibujar elipses, debido a que puede dibujar más figuras, incluyendo el círculo, pero es más compleja.

Drivers de Teclado:

Los drivers de teclado fueron trabajados por parte del kernel. Primero realizando una rutina en ASM que nos permitiera traernos la KEY de la posición del teclado a la que deseamos acceder. Luego en C se realizó un mapeado de que tecla representa cada KEY para un teclado QWERTY. Luego, agregamos una Interrupción en la cual se es llamada cada vez que se recibe un código por parte del teclado y actualiza el valor de la última KEY recibida. Finalmente agregamos una interrupción de software que al llamarla retorna el ascii de la última tecla presionada.

Snake:

Para el desarrollo del juego del snake se terminó implementando de manera tal que cada elemento de la cola sepa cuál es su siguiente. El tablero se guardó en

una matriz de 20x28, conteniendo información de si el casillero estaba ocupado por un jugador y cuál era el siguiente en forma de un struct. Esto se pensó para que intuitivamente se sepa en qué casillero se encontraba cada elemento de la cola de la serpiente, sobre todo a la hora de calcular una posible colisión que cause que el juego termine. A su vez, se crearon variables globales para facilitar el acceso a varios datos que utilizan muchas funciones y no tener que pasarlas por parámetro.

Aunque originalmente la idea era que se refresque toda la pantalla a cada segundo, esto mostraba claros problemas de rendimiento, por lo que se decidió por optimizar la función que se encarga de esto para solamente refrescar el puntaje junto con la posición en donde se encontraba la última posición de la cola, mejorando mucho la fluidez.

Por otro lado, a la hora de generar la nueva posición de la fruta, debido a la imposibilidad de usar la librería estándar de C(donde se encuentran las funciones rand y srand), se tomó la decisión de, en vez de generar un numero random para la nueva posición, se iba a tomar un número de un array de números ya determinados, sin repetir. A pesar de que esto genere repetitividad, al ser 50 los números, esto durante el juego puede pasar más desapercibido.

Por último, se decidieron hacer 2 funciones separadas de “motor de juego”, debido a que se consideró que, a pesar de haber código muy similar, había que implementar una cantidad tan importante de cambios a esta que resultó más práctico hacer una distinta.

Interrupciones de Software:

Para la conexión del kernel y el userSpace, se realizó una de Interrupción de software; la cuales se encargaron de trabajar con los drivers. Estas llamadas permiten trabajar con los drivers de Video, Teclado, Sonido. Esta Interrupción, dependiendo de los parámetros que se le entreguen trabaja con un driver distinto y retorna o modifica los valores requeridos en cada ocasión.
Por parte del usuario, se realizó un solo código en ASM que llama a la interrupción con hasta 5 parámetros mediante la función Syscall().

Excepciones:

En cuanto a las excepciones, se implementó un arreglo de punteros a función para que sea más sencillo identificar de cuales se tratan (por ejemplo, la número 0, cero div, es la de la posición 0).

Por otro lado, para el manejo de excepciones, se implementó una “Blue Screen Of Death”, la cual indica el valor de los registros y cuál fue el error. La forma en la que

muestra el valor de los registros es la misma que la que sirve para mostrarlos incluso cuando se la llama sin necesidad de una excepción. Luego de mostrados estos valores, se indica que se presione una tecla para continuar y volver al sistema. Durante la entrega previa esto no fue posible pero se arregló, reactivando las interrupciones gracias a la instrucción sti. Una vez terminado esto, se regenera el stack por medio de la función getStackBase y se "reinicia" el sistema.

Driver de sonido:

Con la intención de generar un sonido similar al "beep" en el Snake, se utilizó el PIT para lograr acceder al parlante de la PC y así poder setear una frecuencia de sonido a gusto del usuario.
Primero tuvimos que implementar las funciones de "prendido" y "apagado" del parlante, accediendo al port 0x61 del PIT en cada función para controlar su uso.
Al momento de generar sonido, implementamos una función "wait" que espera cierta cantidad de tiempo hasta apagar el parlante. Esta función hace uso del Timer Tick para comparar el tiempo transcurrido con el tiempo deseado de espera, y así obtener un sonido similar al "beep".

Registros:

Los registros se pueden obtener en cualquier momento si se presiona la tecla 'Tab'. Esto muestra una pantalla con todos los valores actuales de los registros. Esto se logra principalmente por una macro en ASM que se encarga de guardar los valores de todos los registros en un array. Luego, desde el kernel Space, se puede obtener este arreglo por medio de un getter que se encuentra en la nativeLib (o sea, la librería exclusiva de kernelSpace), que luego se guarda en una estructura (llamada registerState). Luego, estos se pueden imprimir cuando se ejecuta una excepción o cuando se presione la tecla tab. Para este último, en el administrador de system calls, se modificó la que se encarga de leer para que esta, si es que se obtuvo el carácter con el código ascii 9 (o sea, el tabulador), se despliegue el menú especial que muestra los registros, y otra vez, si se pulsa otro carácter, se sale del menú.

Función Time:

Al momento de obtener la fecha y hora actual se hizo uso del RTC. Para acceder a ella primero se implementó una función en libasm.asm para cada uno de los valores más importantes (hora, minuto, segundo, día, mes y año). Dentro de cada función primero se activa el formato binario para no recibir el dato en BCD, y después se retorna el valor solicitado. Cada valor se guarda en un struct "date" para facilitar su acceso a la hora de imprimir en pantalla.