Instituto Tecnológico de Buenos Aires

INFORME TPE
ARQUITECTURA DE COMPUTADORAS

Integrantes:

- Lopez Vila, Jerónimo (65778).
- Cuneo Gima, Matías (64492).
- Spivak Fontaiña, Federico (60538).

Profesores:

- Valles, Santiago Raul
- Ramos, Federico Gabriel
- Scaglioni, Giuliano

Objetivo:

Implementar un kernel que administre los recursos de hardware de una computadora y muestre las características del modo protegido de intel.

Índice:

1. Introducción

En el presente trabajo se implementó un kernel booteable con Pure64, el cual fue provisto por la cátedra. En este, se separó claramente dos espacios, el user space y el kernel space. Ambos cuentan cada uno con distintas funciones disponibles y sin poder acceder el uno al otro, a menos que sea por los medios provistos para ello, las llamadas al sistema (syscalls). En base a esta última idea, se creó una API con funciones para el usuario, las cuales están declaradas en el manual para dicho usuario.

2. Entorno de Trabajo

Para desarrollar este trabajo se utilizó el editor de texto VS Code, para compilarlo un contenedor de docker con la imagen propuesta por la cátedra “agodio/itba-so:1.0”. Para eso, se ejecuta el comando “compilaTodo.sh”. Por último, para correrlo se usó qemu-system-x86.

El proyecto se corrió en una consola Linux, la cual debió tener instalado docker. Para compilar el proyecto se debió seguir los siguientes pasos:

1. Asignar un directorio al proyecto

2. Correr los siguientes comandos:

```
docker pull agodio/itba-so:2.0
docker run -d -v ${PWD}:/root --security-opt
seccomp:unconfined -it --name <NOMBRE> agodio/itba-so:2.0
```

3. Entrar al entorno del proyecto

4. Correr el siguiente comando:

```
docker exec -it <NOMBRE> bash
```

5. Luego ir al proyecto:

```
cd /root
```

6. Para compilarlo por primera vez se corren los siguientes comandos:

```
cd Toolchain
make all
cd ..
make all
```

7. Para correr el proyecto, ejecutar el programa `'run.sh'`

Luego, para controlar las versiones se utilizó un Git en un repositorio de GitHub

3. Necesidad de separar el espacio

El usuario no debe poder interactuar directamente con el hardware, pues esto sería peligroso para el equipo. No se puede presuponer que el mismo tiene el conocimiento para utilizar el hardware y no dañar funciones importantes. Pero, al mismo tiempo, muchas de las funciones que necesitará el usuario requieren de la utilización de Hardware. Es por esto que es necesario la creación de un espacio que sí pueda interactuar con el Hardware y le provea las funciones que el usuario necesita. Actuando, de esta manera, como un intermediario entre el Hardware y el usuario. A este intermediario, lo llamaremos Kernel Space. Este es solo accesible para el usuario por medio de llamadas al sistema, luego es en este espacio que se recibe la petición del usuario, valida los parámetros y decide que retornar o ejecutar.

4. Funciones del Kernel Space

Como dijimos, el kernel space debe encargarse de ciertas funciones relacionadas con el hardware, por ejemplo: acceder al framebuffer de video, comunicarse con el RTC (real time clock) para obtener información de la hora, comunicarse con el Hardware del teclado, entre otros componentes.

Además, el kernel space debe brindarle todas las herramientas al user space para que este pueda hacer sus funciones. Esas herramientas son las syscalls. Con ellas, el User space puede comunicarse con el Kernel space. Además, se encuentran las interrupciones y excepciones, las primeras siendo rutinas que se ejecutan cuando una pieza de hardware habilitada es utilizada y, las segundas, son aquellas que se corren cuando se registra un error.

4.1 Syscalls

Las llamadas a sistema (Syscalls) son un recurso indispensable para el trabajo práctico, estas son la garantía de que se corran funciones importantes que solo el desarrollador del sistema operativo puede autorizar correctamente. Estas funcionan de la siguiente manera: En código assembler se deben cargar los registros de la manera detallada en la sección de Funciones de User Space y finalizar insertando el comando “int 80h”. Esto envía los parámetros al kernel space y es allí donde se realizan las operaciones especificadas. Las syscalls toleradas por nuestro trabajo son:

- Read (Id 0)
- Write (Id 1)
- getcharNonLoop (Id 2)
- impRegs (Id 12)
- zoom (Id 28)
- sleep (Id 35)
- draw (Id 43)
- screenDetails (Id 44)
- setCursor (Id 45)
- getClock (Id 46)
- playSound (Id 47)
- getMiliSecs (Id 87)

Se eligieron esos Ids para que algunas coincidan con su contraparte en Linux. La primera syscall es Read, la cual devuelve en el buffer la cantidad de caracteres especificados tomados del file descriptor (solo existen tres, entrada estándar, salida estándar y salida por error). La segunda, Write,

escribe los caracteres del buffer a color. La tercera, es similar a read pero no loopea esperando a que se apriete el carácter. La syscall impRegs imprime los registros guardados cuando se apretó la tecla ESC del teclado. Zoom permite aumentar o disminuir el tamaño de la letra. La siguiente permite frenar la ejecución por un número de tics. La syscall draw permite dibujar un pixel del color provisto. La de id 44 provee el ancho y alto en píxeles de la pantalla y la de id 45 permite determinar la ubicación en donde se arrancan a escribir los caracteres. getClock devuelve la hora del sistema, playSound permite emitir sonidos mediante el altavoz interno (PC Speaker), según el índice de sonido y getMiliSecs devuelve la cantidad de ticks contabilizados desde el arranque del sistema.

4.2 Interrupciones y excepciones

El sistema cuenta con rutinas de atención para algunas interrupciones y excepciones. En la interrupt descriptor table (IDT), la tabla donde se cargan las rutinas de atención, se encuentran cargadas las siguientes funciones:

- Timer Tick
- Teclado
- Interrupciones de sistema
- Excepción por dividir por cero
- Excepción por opcode invalido

Las primeras dos son interrupciones de hardware. La de Timer Tick se encarga de registrar un tick más por cada vez que se ejecuta y la de teclado guarda que carácter se presionó al momento de la interrupción en un buffer. La interrupción de sistema es aquella que se corre cuando se ejecuta la instrucción de assembler “int 80h”, la cual ejecuta las syscalls según qué valor recibe en rax. Asimismo, están cargadas dos rutinas de atención a excepciones, la de dividir por cero y la de opcode invalido. Ambas imprimen los registros en el momento en el que sucede la interrupción y cada una luego imprime un mensaje notificando cual fue el error. Finalmente, luego de tres segundos, se retorna a la terminal.

4.3 Modo Gráfico

Para la realización del trabajo, se habilitó el modo gráfico del `x64BareBones` modificando la constante `cfg_vesa`, que viene deshabilitada por defecto. Esto descartó los caracteres predefinidos y la paleta de 16 colores. Por esta razón, fue necesario crear un archivo propio, `font.c`, para obtener los caracteres (8x16 píxeles) y los colores deseados, permitiendo así la impresión de texto y formas requeridas. Para estas funciones, se utilizó el archivo `videoDriver.c` dentro del kernel.

Funciones de Impresión

Para realizar las impresiones en pantalla, se implementó una serie de funciones, entre las que se incluyen:

- Impresión de píxel individual (`putPixel`): permite especificar un color y la posición exacta del píxel en pantalla para cambiar el color de cada píxel según su ubicación.
- Impresión de caracteres individuales (`drawChar`): imprime un solo carácter en pantalla, tomando en cuenta la configuración de color de fondo y de posición.
- Escalado de caracteres (`changeFontScale`): ajusta el tamaño de la fuente mediante una matriz de bits, permitiendo diferentes niveles de escala en tiempo real.

Escritura de Texto

Para facilitar la escritura de texto, se implementaron funciones que permiten mover y gestionar la ubicación del cursor en pantalla:

- `putcharVideo`: permite imprimir un solo carácter, utilizando un sistema que controla el color y la posición del texto. Al llegar al borde de la pantalla, la función gestiona el salto a la siguiente línea. Además, acepta caracteres especiales como el tabulador (`\t`), nueva línea (`\n`) y el borrado de caracteres.
- `printVideo`: escribe cadenas de caracteres en la pantalla, invocando a `putcharVideo` para cada carácter y avanzando el cursor de manera automática.

Control del Cursor

Para gestionar dónde aparece el texto, se implementaron funciones que permiten:

- Cambiar la posición del cursor mediante la función `setCursor`, que especifica coordenadas en pantalla, validando la posición.
- Obtener los detalles de la pantalla con `getScreenDetails`, que devuelve las dimensiones de la pantalla para ajustar el contenido según la resolución.

Escalado Dinámico

Para mejorar el tamaño visual, se añadió la funcionalidad de **zoom**:

- `zoomIn` y `zoomOut`: estas funciones permiten aumentar o reducir el tamaño de la fuente, desde un mínimo de escala de x1 hasta un máximo de x3, acorde al tamaño de la pantalla. Esto ajusta el tamaño de cada carácter en píxeles (8x16, 16x32, 24x48).

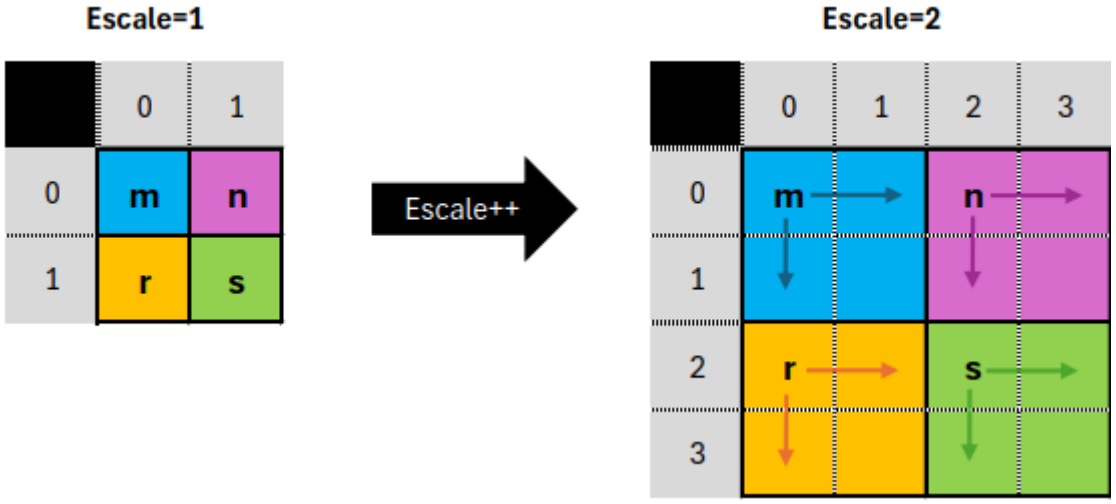

Figura 1: Escalado Dinámico

Esta funcionalidad de zoom fue diseñada para ajustar el tamaño de la fuente en tiempo real, considerándose una manera eficiente en comparación con el uso de varios archivos `font.c` de distintos tamaños. También ofrece la versatilidad de habilitar más niveles de zoom fácilmente, en caso de pantallas de mayor resolución.

Validación de Coordenadas

La función auxiliar `isValidCoordinates` asegura que las posiciones dadas se encuentren dentro de los límites de la pantalla, evitando errores en la impresión de gráficos o texto fuera del área visible.

Font

Como se mencionó anteriormente, se desarrolló la librería de caracteres `font.c`, donde cada carácter se representa mediante una serie de valores hexadecimales. Estos valores están organizados en bloques de 16 bytes de 8 bits, lo que permite que cada bit represente un píxel individual. Si el bit está encendido (1), se imprime un píxel con el color correspondiente; si está apagado (0), se imprime el píxel correspondiente al fondo.

Cada carácter es, por lo tanto, una “matriz” de bits que define su apariencia. Toda esta información está almacenada en un vector `fontBitmap`, que, al solicitar un carácter específico, retorna los bits correspondientes para representarlo gráficamente en pantalla. A continuación, se muestra un ejemplo del carácter `'1'`.

```
0x00,  //........
0x00,  //........
0x18,  //...%%...
0x78,  //.%%%%...
0x18,  //...%%...
0x18,  //...%%...
0x18,  //...%%...
0x18,  //...%%...
0x18,  //...%%...
0x18,  //...%%...
0x18,  //...%%...
0x7e,  //.%%%%%.
0x00,  //........
0x00,  //........
0x00,  //........
```

Figura 2: Bitmap del `'1'`

4.4 Sonido

El PC Speaker es un dispositivo de sonido básico originalmente diseñado para producir sonidos simples como "beeps". Este altavoz se enciende y apaga mediante el uso de puertos, en particular el puerto `0x61`, en combinación con el Programmable Interval Timer (PIT). A través del PIT, el PC Speaker emite tonos de frecuencias específicas alternando entre dos posiciones, `in` (encendido) y `out` (apagado).

Programmable Interval Timer (PIT)

Para emitir sonido, el temporizador 2 del PIT se configura a una frecuencia específica. El PIT genera pulsos eléctricos a la frecuencia indicada, que luego son enviados al PC Speaker, que convierte estos pulsos en sonido. La frecuencia del PIT se establece a través de:

- El puerto `0x43`: que configura el modo de operación del PIT, indicando cómo deben interpretarse los pulsos.
- El puerto `0x42`: que ajusta el divisor de frecuencia, un valor calculado que permite determinar la frecuencia del tono.

De esta manera, `play_sound()` utiliza el divisor para fijar la frecuencia de los tonos generados por el PC Speaker, mientras que `nosound()` apaga el altavoz.

Sonidos Personalizados

El código incluye funciones para controlar secuencias de sonidos específicos, como `soundBeep`, `soundWelcome`, `soundGameOver`, `soundApple`, `soundWinner1`, y `soundWinner2`, entre otras. Cada función genera un tono con una frecuencia y duración definida, permitiendo una salida de sonido personalizada. Esto se logra mediante el uso de `play_sound()` y `nosound()` para el encendido y apagado del altavoz, en combinación con `sleep()`, que define la duración de cada tono.

5. Funciones del User Space

Por otro lado, el User space debe poder interactuar con el usuario. Esto lo logra a través de una terminal de comandos. En la misma, se implementan algunas funciones como: un manual de comandos, poder agrandar o achicar la letra (zoom in y zoom out), una función para ver la hora(clock), otra para mostrar lo que está guardado en los registros(showRegisters), un juego del tipo snake, y una función para cerrar dicha terminal. La terminal funciona de la siguiente manera: Esta contiene un buffer de los caracteres válidos recibidos, estos siendo las letras del alfabeto inglés, los números, tab y espacio, y cuando se apreta la tecla de backspace se elimina el carácter anterior y se corrige el cursor. Una vez se presiona enter, se compara este buffer con los nombres de los comandos previamente mencionados y, en caso de coincidir, se corre el código asociado.

Para lograr esto, en el User Space se usa la interrupción int 80h para generar una syscall que provoque la interrupción deseada. Antes de llamar a dicha interrupción, se debe dejar en RAX el identificador de la syscall que se desea que el procesador ejecute. Además, si dicha syscall necesita recibir algún tipo de parámetro se lo debe enviar por registros en el siguiente orden: RDI, RSI, RDX, RCX, R8 y R9.

Gracias a las syscalls y la interrupción 80h, se logró construir una galería estándar, “stinUser.h”, para manejo de strings, imprimir en pantalla y emitir sonidos. Con la syscall “Read”, se creó getChar, que permitió crear funciones para almacenar strings y caracteres. Con “Write”, putChar y todo lo relacionado a imprimir dichos strings. También se utilizó “Draw” para imprimir pixeles de colores en pantalla. Por último, se utilizó clock para obtener la hora. La librería presenta, entre otras, las siguientes funciones:

- getchar
- putchar
- print
- printColor
- putCharColor
- zoomIn
- zoomOut
- imprimirRegistros
- sleepUser
- clock
- sound
- generarNumeroAleatorio

Todas estas funciones pueden ser de gran utilidad para el usuario, permitiéndole recibir parámetros del teclado, imprimir en pantalla, agrandar y achicar la letra, saber lo almacenado en los registros, mostrar la hora y emitir un sonido.

Un programa que utiliza muchas de estas funcionalidades es el juego del snake. Este programa provee al usuario de dos modos de juego, un jugador y dos jugadores, donde cada “vibora”, una cadena de cuadrados de color, compite por agarrar “manzanas”, cuadrados rojos, dentro de un mapa evitando chocar con si misma o las paredes. En el fondo, cada serpiente es un arreglo de posiciones dentro del mapa y las frutas son coordenadas aleatorias obtenidas con la función de generar un número aleatorio. La estructura de la víbora es esa pues se presupuso que sería la mejor a la hora de volver a imprimirla después de cada movimiento por medio de uso de dos índices, uno para la cola y otro para la cabeza. Además, se cuenta con una estructura de matriz para las posiciones ocupadas del mapa, así se puede verificar rápidamente a la hora de poner una nueva manzana si la ubicación random no está ocupada, por ejemplo. Luego, la interfaz del juego está hecha usando print, getchar, entre otras funciones de entrada y salida y además, a la hora de chequear que se haya introducido un comando válido, las funciones de strings.

6. Conclusiones

En el transcurso del trabajo práctico han habido altercados como bugs difíciles de corregir pero se han sabido superar. Los desafíos más grandes afrontados fueron la organización, el manejo de muchos archivos y la programación de tan bajo nivel. Se desarrollaron muchas habilidades por necesidad, como lo son la búsqueda de información correcta. Visto de que se trató un tema de nicho, fue difícil hallar información correcta y confiable que sea compatible con la consigna y con lo que ya se había trabajado. Finalmente, si bien ha sido un desafío realizar este trabajo, se concluye que se adquirieron nuevos conocimientos, se reforzaron aquellos tratados en la materia y que fue interesante y entretenido programar algunas funciones como por ejemplo, el snake.

7. Bibliografía

- https://wiki.osdev.org/Expanded_Main_Page
- https://wiki.osdev.org/PC_Speaker