Instituto Tecnológico de Buenos Aires

TPE - Informe

72.08 - Arquitectura de Computadoras

Grupo: HomerOS

Integrantes:

Gonzalo Candisano - Legajo N° 62616

Ben Deyheralde - Legajo N° 63559

Emilio Pablo Neme - Legajo N° 62601

Theo Shlamovitz - Legajo Nº 62087

Docentes:

Vallés Santiago Raúl

Merovich Horacio Víctor

Ramos Federico Gabriel

INTRODUCCIÓN:

El objetivo del presente informe es explicar el diseño implementado y las decisiones tomadas con sus respectivos pros y contras.

El Trabajo Práctico Especial busca implementar un kernel para así, administrar los recursos de hardware de una computadora y que el usuario pueda acceder a funciones del Kernel mediante Syscalls. El trabajo práctico consta de dos espacios separados. El primero siendo el kernel space y el segundo el user space. El espacio del kernel interactúa directamente con el hardware mediante drivers a la vez, dándole acceso a funciones al user space así este mismo puede acceder a las funciones sin necesidad de interactuar directamente con el hardware mediante el espacio del kernel.

El trabajo pide implementar una API con el fin de que un usuario pueda acceder a las funciones del kernel. Este acceso se realiza a través de la interrupción de software 80h. Es decir, el user space puede acceder a las funciones del kernel space mediante el uso de *syscalls*.

ENTORNO DE TRABAJO:

Para desarrollar el proyecto, utilizamos el editor de texto Visual Studio Code. Para compilar el proyecto, utilizamos un contenedor de Docker creado con una imagen provista por la cátedra llamada *"agodio/itba-so:1.0"*. Y, para correr el proyecto, utilizamos la herramienta qemu-system-x86.

Para poder trabajar a la par como por separado, creamos un repositorio en GitHub donde cada uno "subía" (commit + push) los cambios que realizaba. Igualmente para trabajar como grupo y que todos tengamos un conocimiento de todas las partes del proyecto, la mayor parte del mismo fue realizada por los 4 en llamada en Discord turnandonos para programar mientras se compartía la pantalla del que estaba programando para poder todos colaborar a pesar de no ser el que está escribiendo en el momento.

Por último, como base para el proyecto utilizamos la base provista por la cátedra del repositorio de BitBucket: https://bitbucket.org/RowDaBoat/x64barebones/wiki/Home.

DESARROLLO:

El kernel space debe manejar los recursos del hardware y controlar el acceso del user space a los mismos por lo que sus tareas no deben superponerse. Como por ejemplo, el manejo de interrupciones (responsabilidad del kernel space) y la terminal de comandos (responsabilidad del user space).

Para poder acceder al user space, la siguiente línea de comando fue escrita dentro del archivo *kernel.c* que redundantemente, se encuentra en el kernel space:

((EntryPoint)sampleCodeModuleAddress)();

Para acceder al kernel space desde el user space, obtuvimos de la Interrupts Descriptor Table, la interrupción 80h. Utilizando el registro *rax* como identificador, se accede a las distintas *syscalls*. Los parámetros recibidos por cada syscall son recibidos utilizando los mismos registros que utiliza C en 64 bits: *rdi, rsi, rdx, rcx, r8 y r9*.

Dentro del user space, se encuentra la implementación para el funcionamiento de la terminal de comandos y, el funcionamiento de los mismos.

Estos comandos son los que le permiten al usuario interactuar con el sistema operativo. Algunas funciones de la librería estándar de C también fueron implementadas tal como pedía la consigna provista por la cátedra.

El kernel provisto por la cátedra se encontraba en modo texto por lo que se debía modificar el valor de la dirección correspondiente al lugar en donde se quería imprimir siempre dentro de la memoria correspondiente al video/pantalla. En un principio empezamos a investigar cómo cambiar el proyecto a modo video ya que era lo que se pedía en la consigna y pudimos realizarlo gracias a una clase práctica que dio el ayudante Gianfranco Magliotti explicando que se realiza cambiando la siguiente linea de codigo de 0 a un 1:

cfg_vesa: db 1 ; By default VESA is disabled. Set to 1 to enable.

Al activarse el modo video, para imprimir en pantalla se debe realizar pixel por pixel. Para lograr el funcionamiento, implementamos un driver de video (*videoDriver.c*). Para la impresión de caracteres implementamos una “font” (*font.h*) la cual consiste en una matriz de caracteres la cual indica, según el caracter, que pixel debe estar prendido y que pixel apagado.

En el user space (*Userland*) se encuentran archivos los cuales, en *sampleCodeModule.c* inicializa el UserSpace*, shell.c* que realiza el funcionamiento de la terminal leyendo comandos y ejecutandolos, y *pong.c* que implementa el pong (*pong.c*). El manejo de strings se encuentra en *uStrings.c*.

El kernel space (*Kernel*) principalmente se encarga de manejar el driver de video (mencionado anteriormente), el driver de teclado, el de sonido, interrupciones y excepciones.

En cuanto a las interrupciones, son realizadas insertando entradas de la Interrupt Descriptor Table. Dicho cargado se realiza en *idtLoader.c* y la definición de las rutinas al momento de que se realice una interrupción en *interrupts.asm.*

Si bien algunas de las interrupciones ya estaban realizadas para el PRE-TP, agregamos algunas más las cuales eran necesarias.

La interrupción de división por cero (0x00) y la de codigo de operacion invalido (0x06) en *irqDispatcher.c,* las cuales se manejan a través de *exceptions.c* que al ocurrir alguno de estos tipos de errores le dará su funcionalidad. La interrupción encargada de las syscalls (int 80h) son derivadas al handler de las mismas el cual se encuentra en *syscalls.c* para actuar con su debido funcionamiento.

En cuanto a las syscalls, se encuentran tanto en el user space como en el kernel space ya que para que el user space se pueda comunicar con el kernel tiene que llamar a una syscall ubicada en *usyscalls.c,* la cual carga los valores (en cada syscall son valores distintos dependiendo lo que se necesita en cada una pero, siempre el primer valor va a ser el id) en los registros, utilizando la función *sys_call* la cual se encuentra en *syscalls.asm* en *Userland* la cual llama a la interrupción int 80h. Luego, ya en el kernel space, *interrupts.asm* detecta la interrupción llamando a *syscallHandler* ubicado en *syscalls.c (*en *Kernel),* el cual según el id recibido llama a la función que le corresponde a la debida syscall así, realizando su debida función

Primero creamos las syscalls encargadas de leer y escribir (*sys_read* y *sys_write*) y algunas variantes de las mismas como por ejemplo *sys_write_color* para poder agregarle un color a lo que se escriba en pantalla. Cabe destacar que para una óptima forma de utilizar los file-descriptors los definimos como STDIN (0), STDOUT (1) y STDERR (2).

Luego, en algunas de las funcionalidades, se debía poder observar el contenido de los registros en un momento en específico (pulsando la tecla *CTRL* se realiza un "screenshot" de los mismos), para esta funcionalidad creamos la syscall *sys_get_registers* (para corroborar el funcionamiento de la misma, se creó la función *fillregs* la cual lo que haces es ir decrementando los registros uno por uno así al tocar la tecla *CTRL* luego de ejecutar el comando, luego al imprimir los registros se puede ver como quedan en un orden descendiente).

Luego creamos las syscalls correspondientes a obtener tanto la hora como la fecha del sistema operativo accediendo al RTC (*sys_get_time* y *sys_get_date*), las cuales ya teníamos la mayor parte resuelta ya que lo habíamos realizado para el PRE-TP.

Algunas de las demás syscalls creadas a destacar son la *sys_ clear_screen* la cual habilita que al escribir el comando *clear* se "borre" toda la pantalla, y *sys_play_sound* así pudiendo armar las "canciones" que se encuentran en el trabajo tanto como los sonidos del pong.

Finalmente, para el correcto funcionamiento del pong, se necesitaron agregar una serie de syscalls como por ejemplo *sys_play_sound* mencionado anteriormente, *sys_get_screensize* que permite ejecutar correctamente el pong sin importar el tamaño de la pantalla en el cual se encuentre, *sys_draw_rect* que se encarga de dibujar en la pantalla los componentes necesarios para que funcione el pong y, *sys_get_ticks* que devuelve los ticks actuales del sistema y así poder realizar lapsos de tiempo dentro del videojuego.

CONCLUSIÓN:

El trabajo nos obligó a investigar y profundizar nuestros conocimientos sobre las interrupciones, las syscalls y la separación del user space con el kernel space. Fue un trabajo con muchos obstáculos, pero trabajando en equipo consideramos que pudimos superarlos. A pesar de que estos obstáculos fueron frustrantes en sus momentos, al terminar el trabajo podemos decir que son estos mismos los que nos forzaron a adquirir nuevos conocimientos.

Habiendo ya cumplido con la consigna del trabajo, implementamos algunas funciones extra como por ejemplo las canciones utilizando el sonido de la computadora con distintas frecuencias o mismo como representar imágenes pixeladas en el entorno de trabajo.

Por último, nos quedamos con las ganas de seguir implementado funciones extra que por tiempo no llegamos (hay un easter-egg en el trabajo que no se encuentra en la lista de comandos) pero, lo más importante es que como equipo, disfrutamos el trabajo ya que a pesar de que por ciertos momentos fue frustrante, pudimos resolver los obstáculos. Consideramos que como grupo trabajamos bien entre los 4 y adquirimos nuevos conocimientos.