Instituto Tecnológico de Buenos Aires

Informe

72.08 - Arquitectura de Computadoras - TPE

Integrantes:

Gianna Lucia Nacuzzi - 64006
gnacuzzi@itba.edu.ar

María Agostina Squillari - 64047
msquillari@itba.edu.ar

Pilar Frutos - 64225
pfrutos@itba.edu.ar

1. Introducción

Este Trabajo Práctico Especial consiste en implementar un kernel booteable con Pure64, provisto por la cátedra, que administre los recursos de hardware de una computadora y muestre características del modo protegido de Intel. El mismo debe proveer una API para que aplicaciones de usuarios puedan utilizar estos recursos.

2. Entorno de Trabajo

Para llevar a cabo el desarrollo del proyecto, empleamos el editor de código VS Code. En cuanto a la compilación del proyecto, utilizamos un contenedor Docker. El proyecto se desarrolló en un repositorio de GitHub, lo que permitió trabajar de manera colaborativa desde nuestras respectivas máquinas y mantener un control de versiones adecuado.

Por un lado, dos integrantes cuentan con Windows como sistema operativo en sus computadoras. Se utilizó WSL (Windows Subsystem for Linux) versión 2 para poder correr en el entorno de Linux. Docker debe estar instalado en Windows y configurado para utilizar el “WSL2-based engine”.

Por otro lado, la otra integrante cuenta con una Mac con chip Intel. Para ejecutar el proyecto, se descargó la aplicación Docker Desktop. Además, se utilizaron las utilidades de Homebrew para instalar los diversos programas necesarios para compilar el código. Todo el proceso se llevó a cabo en la terminal de una Mac.

Los comandos para compilar el proyecto son los siguientes:

```
docker start tpearqui
docker exec -it tpearqui make clean -C/root/Toolchain
docker exec -it tpearqui make clean -C/root/
docker exec -it tpearqui make -C/root/Toolchain
docker exec -it tpearqui make -C/root/
docker stop tpearqui
```

donde tpearqui es el nombre del contenedor elegido o creado previamente para correr el proyecto.

Por último, es posible que al intentar hacer el run.sh este no tenga permisos de ejecución. Es necesario entregarlos para correlo.

3. Separación Kernel - Userspace

El kernel se encarga de interactuar directamente con los recursos del sistema y ofrece una interfaz de acceso a estos recursos para los programas que se ejecutan en el user space. Por lo tanto, es esencial mantener una clara separación entre el kernel y el userspace.

En consecuencia, el kernel no está limitado al user space desarrollado en este proyecto. El kernel está programado para ejecutar cualquier otro módulo de userspace. En contraparte, el user space si está ligado al kernel, dado que el mismo le proporciona las funciones para poder interactuar con el hardware.

El Userland puede acceder a las funciones del Kernel a través de una interfaz de programación de aplicaciones (API) proporcionada por el mismo. Este acceso fue implementado a través de la interrupción de software 80h, ya que estos dos módulos se encuentran en distintos espacios de memoria. Además, se han establecido un conjunto de funciones para interactuar con la API mencionada. Esta API está basada en las bibliotecas de C, contando con funciones como scanf, printf y putchar.

4. Syscalls implementadas

Basándonos en el ejemplo de Linux, se ha designado la posición 0x80 en la IDT (Tabla de Descriptores de Interrupción) para las llamadas al sistema (syscalls) del sistema operativo. En esta posición, se encuentra un puntero que apunta a una función escrita en lenguaje assembler, la cual se encarga de preparar los registros según las convenciones de C. Se ha optado por mantener el número de id de las syscall en el primer argumento, seguido de los argumentos restantes en el orden en que fueron enviados. Una vez que los argumentos (es decir, los registros) han sido posicionados, se invoca a la función syscallDispatcher, implementada en el archivo syscalls.c, la cual ejecuta las rutinas correspondientes a cada llamada al sistema según su identificador.

Fue necesario tener en cuenta los casteos necesarios a la hora de utilizar un argumento dado que como se envió a través de registros todos eran del tipo uint64_t. Por lo tanto, se requirió realizar los castings adecuados para asegurar la correcta asignación de los valores.

Un problema con el que nos encontramos fue la hora del sistema, la hora que se imprime en pantalla es la de GMT.

Por último, con respecto a la syscall de make_sound tuvimos un inconveniente al intentar hacer el sonido con computadoras con sistema operativo distinto a macOS. Debido a que aunque se realizaron los cambios adecuados para activar el módulo de sonido en el respectivo run.sh nunca se logró que realice el sonido al llamar a la función. Además, agregar los flags necesarios en el run.sh generaba la aparición de una lista de advertencias, por eso se tomó la decisión de solo activar el módulo del sonido para las computadoras con macOS.

5. Manejo de interrupciones

Por un lado, la gestión de interrupciones se lleva a cabo mediante la inserción de entradas en la IDT (Tabla de Descriptores de Interrupción), la cual comienza en la dirección 0x00. El proceso de carga de esta tabla se realiza en el archivo idtLoader.c, mientras que las rutinas de interrupción están definidas en interrupts.asm.

Por otro lado, las excepciones son todas enrutadas hacia una función exceptionDispatcher(...) definida en exceptions.c, a la que se le pasa el número identificador de la excepción que ocurrió. Esta muestra un mensaje de error en la pantalla junto con el instruction pointer y un dump de los registros en el momento que se produce el error. Una vez que la excepción ha sido tratada, el sistema retorna al Shell.

El kernel está preparado únicamente para recibir las excepciones Divide by Zero y Invalid Opcode.