Informe: Trabajo Práctico Especial

Integrantes y legajo:

Henestrosa, Augusto	59189
Letterio, Francisco	58408
Barbieri, Guido	59567
Choi, Francisco	59285

Introducción:

El trabajo práctico consiste en la sincronización entre los programas del userspace y el kernel space. Para dicho objetivo, se creó un sistema de interrupciones descrita a continuación a través de una tabla IDT. Los drivers del hardware y los syscalls fueron implementados para poder interactuar con el userspace de modo que no puedan acceder mutuamente los dos espacios ya mencionados, ya que el usuario no debería poder manipular aspectos del kernel space para asegurar el correcto funcionamiento del sistema operativo.

IDT

La IDT estuvo mayormente basada en la provista por la cátedra. Se armó un dispatcher al que todas las interrupciones llaman con el primer parámetro siendo el de que interrupción es. El mismo luego llama a acciones que hacen lo que la interrupción necesita. Para la interrupción 80h se armó otro dispatcher distinto a la del resto de las interrupciones dado su formato. La misma estuvo ligeramente basada en la int 80h de linux pero a diferencia de esta,nosotros pasamos los parámetros a través de los mismos registros por los que recibe parámetros una función en c(rdi,rsi,etcétera, en vez de usar rax,rbx,etcétera). Esto se hizo simplemente por comodidad de uso.

Manejo de excepciones

En cuanto a las excepciones solo se cargaron,por nosotros, dos de ellas a la IDT, la excepción 0(division by 0) y la 6(invalid opcode). Las mismas se pusieron en su lugar correspondiente en la IDT(la posición 0 y 6 respectivamente). Para su implementación se usan tres funciones, una que imprime el nombre y valor de un registro, dados el valor del registro y cual es el mismo, otra que imprime información sobre la excepción y por ultimo, una que reinicia la cpu. Esto se decidió ya que es la forma más sencilla de salvarse de las mismas y volver a la terminal(dado que allí inicia). Previo a este reinicio se espera a que el usuario presione una tecla(de esta forma el mismo tiene tiempo de leer los valores de los registros).

PIC

El pic se configuró en la función loadIDT del kernel para que el mismo permita la IRQ0 e IRQ1. Las mismas son la interrupción del pit y la interrupción del teclado, respectivamente. Estas dos son las únicas dos interrupciones que necesitamos habilitar

Driver de video

El bitmap de los caracteres fue sacado de
https://github.com/dhepper/font8x8/blob/master/font8x8_basic.h

El driver provee distintas funciones para dibujar a pantalla, así como escribir un caracter o un string. Además provee la opción de hacer esto último como consola para minimizar la actualización de píxeles. Lleva registro del cursor y ofrece una función para manejo arbitrario del mismo. Esto en realidad tendría que ser un manejo exclusivo de la aplicación, mientras que los drivers tendrían que ofrecer las funcionalidades mínimas. Sin embargo, el error fue percibido demasiado tarde, por lo que no pudimos delegar estas tareas al intérprete a tiempo.

Driver de teclado

Para el driver de teclado se eligió el siguiente funcionamiento.

- Se utiliza un mapeo para los códigos estándares del teclado americano del archivo keyMap, para mapear al ascii correspondiente.
- Se utiliza un buffer circular de máximo 100 caracteres.

A partir de una interrupción de teclado se llama a la función keyboard handler.
Dentro de esta función se chequea en primer lugar si el código recibido está dentro del límite de las teclas presionadas, es decir si la interrupción fue a partir de la presión de una tecla.
Luego, se verifica si es una de las acciones especiales(ctrl, shift, left shift of CAPS). En caso que sea de una de estas acciones se activan los flags correspondientes.
En caso que no sea una de las acciones especiales, entonces verificamos si hay algunas de las acciones especiales activadas, aquí hay varias alternativas: si por ejemplo está activada la el shift entonces obtenemos la el char correspondiente utilizando el mapa del shift activado.
En caso que no haya ningún flag activado y sea simplemente un char válido, se agrega al buffer tras obtener su caracter correspondiente en ascii.

Para obtener datos sobre el buffer del teclado se utilizan las siguientes funciones:

- Getchar: obtiene la primera letra presionada que está presente en el buffer y la quita del mismo, devuelve 0 si está vacío.(No utilizada en la implementación).
- Get last input: Devuelve el último carácter ingresado en el buffer y lo remueve de la posición. En caso que esté vacío devuelve 0.

Driver de sonido

Para el manejo de sonido se usa el speaker de la pc. Para el manejo del mismo encontramos un código en internet que nos permite mandar al mismo un código y que el mismo suene a una cierta frecuencia acorde al código enviado. Del mismo lugar obtuvimos también una tabla que mapea estos códigos a sus respectivas notas musicales. Si bien no

se utilizó en el trabajo(esto es la utilidad de tener las notas musicales mapeadas al código que se debe pasar), se barajó la idea de que el juego tocara una canción cuando iniciaba sin embargo, en el final, esto no se hizo. Por otro lado, la función se utilizó(a través de un syscall) para los distintos sonidos que se ejecutan durante el juego(cuando se agranda la viborita y cuando se pierde).

Interrupt 80

La interrupción 80 es el nexo entre el userspace y el kernel space.

En primer lugar cuando se requiere acceder al kernel space a traves del userspace, para utilizar alguna función de la librería estándar implementada en el proyecto se llama a una interrupción 80 mediante una syscalls.
Cada sys call tiene una función en assembler para el pasaje de parámetros y el retorno de un valor en caso que lo requiere. Para el pasaje de parámetros se utilizó el pasaje de parámetros por defecto, a los registros, es decir: el número de la sycalls se pasa en rdi, y los argumentos 1, 2 y 3 se pasan en rsi, rdx y rcx respectivamente.
Una vez que se recibe la interrupción se llama a una función en assembler llamada interrupt 80, que además de realizar todo lo necesario para que funcione la interrupción llama a un interrupt80 Action dispatcher implementado en C.
Esta función funciona mediante un switch que en base al número de syscall llama a alguna de las funciones de las syscalls implementadas en el kernel.

Las decisiones en torno a qué número asignarle a cada syscall se tomaron en un principio basado en las syscalls de linux, sin embargo , a medido que progreso el trabajo nos alejamos de este modelo debido a las grandes diferencias entre nuestras necesidades y las syscalls de linux. A partir de entonces, la elección fue mayormente un tema de comodidad.
El orden de las syscall basa en:

- 0-10: Salida, es decir syscalls que imprimen a pantalla y sonidos.
- 10-20: Entrada, syscalls que obtienen información del kernel.
- 20-30: Otros

Terminal

La terminal consiste en un while(1) infinito que constantemente solicita información del teclado o ejecuta un comando.
El proceso de la terminal consiste en 3 pasos:

1. En primer lugar, ejecuta readNewInput que se encarga de almacenar internamente potenciales comandos en un buffer y de imprimir la información tipeada en pantalla.

2. En caso que se presione la tecla Enter, se ejecuta handleCommand, que compara la información del buffer con los comandos disponibles y los ejecuta en caso que sea un comando válido.
3. Luego de ejecutar el comando(o no) se limpia el buffer y se imprime una nueva línea de terminal.

El manejo de comandos es básicamente una estructura constituida por un descriptor y un puntero a la función por cada comando disponible en dicha aplicación.

Juego snake

Para el juego snake es necesario utilizar los drivers de video, sonido, teclado, y otras funciones que acceden al kernel para obtener la hora y la cantidad de ticks del PIT. El período más corto de tiempo a manejar es un tick del PIT, por lo que la máxima velocidad de la víbora como el mínimo tiempo de duración de un sonido se ven limitados por un tick del PIT como mínimo intervalo de tiempo.
Al no disponer de memoria dinámica, el juego almacena una matriz gigante desde el principio. Además por el mismo motivo, quedan hardcodeadas algunas variables como el tamaño de la pantalla (1024 * 768, tamaño que permite el modo VESA), ya que es necesario tenerla en tiempo de compilación y no en runtime. El juego en sí está optimizado al máximo en cuanto a actualizaciones de pantalla, tocando únicamente el mínimo indispensable de píxeles.También se maneja un buffer de input del jugador para controlar movimientos “ilegales” (cambiar de dirección en 180º) pero facilitar movimientos rápidos (apretar dos teclas rápidamente).

Hubo un error de último momento con el manejo del tiempo de la snake. Se hacía con el RTC de la pc, pero por el error se decidió cambiar el manejo del tiempo a través del PIT, por más que resulte menos certero en cuanto al cálculo del tiempo de juego.

Fuentes

La información utilizada tanto para el manejo de excepciones como para el armado del idt en general se sacó mayormente del ejemplo de la cátedra y de https://wiki.osdev.org.
Tambien se uso como fuente un pdf de una catedra de la Facultad de Exactas de la Universidad de Buenos Aires (https://campus.exactas.uba.ar/pluginfile.php/93694/mod_folder/content/0/interrupciones-en-modo_published.pdf?forcedownload=1). El mismo se usó para profundizar en cómo se arma el stack previo al llamado de una excepción.
Ademas de esto, se uso http://stanislavs.org/helppc/ (junto con el previamente mencionado https://wiki.osdev.org) en el transcurso de todo el trabajo
Para el manejo de sonido se uso mayormente el siguiente link como informacion: http://muruganad.com/8086/8086-assembly-language-program-to-play-sound-using-pc-speaker.html