

**Arquitectura de Computadoras**

**TPE**

Franco Baliarda - 58306

Facundo Astiz - 58333

Miguel Di Luca - 58460

**Análisis del proyecto**

El trabajo práctico consiste en un bootloader y un pequeño sistema operativo. Este está subdividido en las carpetas Kernel, Userland, Bootloader, Toolchain e Image, aunque el grueso del código se encuentra en las primeras dos.

Las carpetas Kernel y Userland representan el *kernelspace* y el *userspace* respectivamente. Ambas contienen una carpeta *asm* con los códigos en Assembler, una carpeta *include* con los archivos .h necesarios, y por último fuera de estas dos carpetas se encuentran los archivos en código C.

*Kernel*

Como ya se dijo, la carpeta Kernel consiste en representar el *kernelspace*. Aquí están todos los archivos y código necesario para comunicarse con el hardware, que luego el usuario podrá usar por medio de system calls. Los archivos de C que se encargan del manejo de hardware son *keyboard\_driver.c,* *video\_driver.c y sound\_driver.c*. El primero de estos es el controlador de teclado, que registra las teclas por medio de interrupciones y las escribe luego en el buffer de entrada estándar. Para obtener el código de la tecla que se presionó utiliza el archivo *keyboard.asm*, que escribe y lee en los puertos del teclado. El segundo es el controlador de video que se encarga de ofrecer funciones para el manejo de pixeles en la pantalla (dibujo de pixeles, su color, restaurar la pantalla, realizar un backup de la pantalla, etc.). Para lograr esto se debió habilitar el VESA desde la carpeta Bootloader. Luego se tiene los archivos llamados beep.asm y rtc.asm, donde el primero se encarga de activar y desactivar el sonido del PC Speaker mientras que el segundo se encarga de las interrupciones de hardware causadas por el rtc y da la funcionalidad de un temporizador con el cual se puede coordinar la ejecucion de una funcion luego de un tiempo dado. Por ultimo se encuentra sound\_driver.c que combina las funcionalidades de beep.asm y time.c para lograr reproducir un pitido de corta duracion.

El manejo de interrupciones utiliza varios archivos C y Assembler. Primero, está el archivo *idtLoader.c* cuya función es cargar las entradas de la IDT para que las interrupciones del teclado o las syscalls por ejemplo funcionen. Para hacer esto utiliza una estructura para representar un descriptor de la tabla, y un puntero a esta estructura con valor 0 (esta es la posición donde está cargada la IDT). Cuando se carga cada entrada, se le pasa el puntero a la rutina de interrupción para que esta luego sea ejecutada. Las rutinas de interrupción, ya sean de hardware o de software, se encuentran en el archivo *interrupts.asm*. En el caso que las interrupciones sean de hardware, las rutinas llamaran al irqDispatcher en *irqDispatcher.c* con el número de IRQ que identificará que función correr. Con las excepciones ocurre un proceso similar pero llamando al exceptionDispatcher en *exceptionDispatcher.c*, pasándole como parámetro el número de la excepción a ejecutar. También se le pasa el valor del instruction pointer cuando ocurrió la excepción (se obtiene desde el stack), al igual que un puntero al stack luego de hacer un push de todos los registros para que pueda después escribir estos valores por la salida de error. Después de terminar de ejecutar la excepción, se vuelve a main de Userland para que la consola vuelva a poder ejecutarse. Para lograr esto se guardan dos punteros, uno que apunta a la dirección en memoria donde está el sampleCodeModule (el código main de Userland), y otro que apunta a la dirección en memoria en donde empezar el stack cuando se comienza a correr el código de user. Estos valores se pushean al stack al finalizar el código del exceptionDispatcherpara que cuando se ejecute la instrucción *iretq* los levante y realice la operación de retorno. Por último, está la interrupción 80h que corresponde a las system calls. La rutina de interrupción llama al syscallDispatcher en *syscallDispatcher.c* que ejecutará la función correspondiente dependiendo del número de syscall pasado como parámetro. Todos los dispatchers que se dijeron contienen un vector con los punteros a las funciones a ejecutar, de esta manera se gana eficiencia ya que el número que se pasa como parámetro actúa como índice del vector.

Finalmente, hay un archivo más llamado *std\_buffers.c* cuya función es cargar los buffers cuando arranca el kernel, y brindar funciones para escribir y leer en estos. Los buffers son la entrada estándar, salida estándar y la salida de error. Todos estos tres buffers tienen un tamaño fijo y son cíclicos, es decir, cuando se llega al final entonces vuelve al principio. De esta manera, se puede ir escribiendo y leyendo sin tener que preocuparse por el espacio que queda en el buffer. Si el buffer se llega a llenar por no ser leído, entonces empezará a escribir por sobre lo primero que se escribió.

Todos estos archivos funcionan en conjunto cuando se empieza a ejecutar el *kernel.c*. Este tiene una función main donde se carga la IDT por ejemplo, y luego pasa a ejecutar el main en Userland.

*Userland*

Por el otro lado, la carpeta Userland representa el *userspace*. La función principal en esta carpeta es dar al usuario funcionalidades para interactuar con el Kernel. Para esto se creó la función *\_syscall* en *syscall.asm* que realiza una int 80 con los parámetros necesarios.

Para que el usuario pueda interactuar con el Kernel y el código se utiliza una consola en el archivo *console.c*. Esta consola se inicializa y arranca cuando empieza a correr el código en Userland. La consola lee de entrada estándar y luego con esa entrada el usuario podrá ejecutar diferentes comandos o módulos. Cuando se ejecuta algún comando, la consola luego lee del buffer de salida estándar y salida de error para imprimirlo en pantalla y dar al usuario cierto feedback. Para imprimir en pantalla se creó un *screenManager.c* que se encarga de hacer diferentes system calls para escribir o borrar cosas sobre la pantalla. Para poder escribir caracteres se utiliza el *font.c* que contiene un bitmap de todas las letras imprimibles, que indica que pixeles dibujar sobre la pantalla. Se eligio incluir funciones como writeChar en el userland para permitir al programador mayor customizacion y mantener el kernel lo mas pequeño posible y con funciones genericas e indispensables.

Para ejecutar los diferentes comandos, la consola hace uso del *commandDispatcher.c* que se encarga de ejecutar los diferentes programas dependiendo de que se haya escrito por el buffer de entrada estándar. Por ejemplo, el comando de ayuda es *man* que nos muestra un manual con los diferentes módulos y su funcionalidad. Una vez que el comando solicitado por el usuario es ejecutado, se vuelve a la consola para que imprima lo que haya escrito el programa por salida estándar o de error. En el caso de que el comando no se reconozca, escribirá por salida de error que el comando no se pudo ejecutar. Algunos módulos se ejecutan desde el mismo archivo ya que son más triviales pero otros no. Para el reloj digital por ejemplo, se utiliza el archivo *graphicClock.c* que cumple la función de dibujar el reloj en pantalla, actualizarlo cada vez que pase un segundo y dar al usuario ciertas funcionalidades como cambiarlo de color mientras reproduce un sonido o saimplemente cambiar de tamaño las letras tamaño. Para las excepciones por otro lado, está el archivo *exceptionTester.asm* que contiene las rutinas en Assembler con las instrucciones necesarias para hacer saltar la excepción que se desee.

Por último, están los archivos *stdio.c* y *stdlib.c* que cumplen la función de librerías para los archivos de C que usamos. El primero contiene funciones relacionadas con entrada y salida como *printf*, *getchar, putchar* y *scanf*. El segundo contiene otras funciones útiles que facilitan ciertas operaciones como *intToChar* o *strcmp*.

*Referencias*

<https://wiki.osdev.org/Main_Page>

<http://muruganad.com/8086/8086-assembly-language-program-to-play-sound-using-pc-speaker.html>

https://stackoverflow.com/