Informe

T.P.E. Arquitectura de las Computadoras

I.T.B.A. Q1 2015

Autores:

- Matías Nicolás Comercio Vázquez (55309)
- Juan Moreno (54182)

Índice

Introducción

Este sistema operativo fue desarrollado como un trabajo práctico para la materia Arquitectura de Computadoras, dictada en el ITBA en Q1 2015.

El sistema está implementado para una arquitectura Intel de 64 bits en Long Mode.

Al iniciar el sistema, se ejecuta un bootloader, provisto por la cátedra mencionada anteriormente, que deja al sistema en el modo long.

Una vez en este estado, cada grupo de alumnos tuvo que desarrollar su propia implementación de sistema operativo, que respondiese a las siguientes consignas:

a. Contar con un intérprete de comandos - en modelo de terminal - , que demuestre el funcionamiento del kernel, al tomar los comandos por entrada estándar y mostrar el resultado por salida estándar.
b. Funciones:
    i. Visualizar la hora del sistema.
    ii. Cambiar la hora del sistema.
    iii. Ayuda: mostrar los distintos programas disponibles.
c. Poseer un salvapantallas de tiempo configurable, que se desactive cuando se detecte alguna interacción con el sistema.
d. Agregar cualquier otra funcionalidad que el grupo considere apropiada para mostrar el funcionamiento interno del sistema.

Con estos objetivos en mente, este grupo desarrolló la siguiente estructura de proyecto.

Estructura interna

El proyecto está dividido en 3 grandes secciones:

- Bootloader (Pure64: provisto por la cátedra).
- Kernel Land.
- User Land.

Kernel Land

Este espacio se subdividió de la siguiente manera: los archivos fuentes programados en C fueron ubicados en la carpeta *src* mientras que los programados en assembly en la carpeta *asm*. Los includes se ubicaron en la carpeta *include*. Los drivers son una subdivisión de *src* y se ubican en la carpeta *drivers* dentro de la misma.

Los archivos principales del Kernel se ubicaron en su carpeta principal ya que son aquellos indispensables para arrancar. Estos son: *kernel.c*, *kernel.ld*, *loader.asm* y *moduleLoader.c*.

El archivo *define.h* (ubicado en *include*) contiene las definiciones generales y tipos de datos que se utilizan a lo largo de todo el Kernel.

Dentro del Kernel se presentan las siguientes funcionalidades:

Drivers

Video (video.c):

Implementa las funciones para interactuar directamente con la VGA en modo texto. En este modo se cuenta con un screen de 25 filas por 80 columnas y un cursor. Cada elemento de la matriz representa un caracter del screen y el mismo está subdividido en 2 partes: la primera es la del valor del caracter y la segunda la del atributo que representa el color de texto y de fondo.
Algunas de las funcionalidades de este driver son:

- Habilitar y establecer la posición y forma del cursor.
- Escribir caracteres en pantalla en la posición actual del cursor sin la necesidad de actualizarlo explícitamente (ya que lo hace automáticamente).
- Escribir caracteres en una posición deseada dentro de la pantalla sin que se actualice el cursor.
- Imprimir saltos de línea.
- Asegurar que siempre se escriba dentro de los límites de la pantalla.
- En caso de llegar a completar la pantalla, mover todas las filas para arriba (borrando la primera), generando así una nueva línea para escribir.
- Permitir conocer el valor de un elemento en una posición de la pantalla.

- Permitir establecer el color de fondo y de texto con el que se van a escribir los próximos caracteres. Asimismo permite cambiar el color de fondo y de texto de todos los caracteres actuales (ya existentes) en pantalla.

Keyboard (keyboard.c):

- Permite procesar los códigos recibidos (scan codes) a partir de las interrupciones generadas por el teclado, y transformarlas en caracteres *imprimibles* o *funcionales*.
- Una vez que el código es procesado, se le informa al manager que ha llegado una interrupción de teclado, y se le envía la tecla ya decodificada, para que pueda realizar la operación pertinente.
- Brinda la posibilidad de almacenar caracteres en un buffer de tipo circular, para un mejor aprovechamiento del espacio, y por ser una implementación más versátil que la del buffer lineal.

Timer Tick (timertick.c):

- Permite informar al manager la recepción de la interrupción del timer tick.

RTC (rtc.c):

- Permite leer y establecer los valores relativos a la fecha y hora del sistema.

Manager (manager.c)

Contiene todas las funciones que permiten mediar entre las funcionalidades brindadas al usuario y las implementadas por el kernel.

Las funciones que contiene permiten:

- Recibir las teclas interpretadas desde el teclado, a partir de lo cual, debe decidir qué hacer con ellas: si tienen alguna acción asociada, ejecutarla; si son imprimibles, y la impresión está habilitada, imprimirlas; decidir si es necesario que las teclas se guarden en el buffer del teclado; deshabilitar el salvapantallas en el caso de estar activado.
- Decidir qué hacer frente a la llegada de una interrupción del timer tick: en líneas generales, actualiza contadores que son utilizados como control para ejecutar otras acciones a cabo de un tiempo programable (en nuestro caso, el salvapantallas).
- Recibir las solicitudes de escritura en la salida estándar, mediando entonces (pero no directamente) entre la interrupción generada desde el User Space y el driver de video.
- Administrar el uso de las diferentes terminales.
- Administrar el estado del salvapantallas.

Syscalls (syscallHandlers.c)

Contiene, entre otras, una función (*syscallHandler*) que, en base a un código, elije qué otra función debe ser ejecutada a continuación. El código recibido por la función es enviado

desde el User Space, a través de la int 80h, y recibido en el Kernel Space por la rutina de atención de interrupción de la int 80h.

Interrupts (interrupt.asm)

Contiene las rutinas de atención de interrupción seteadas en la IDT, entre las que se encuentra la rutina de atención de la interrupción int 80h.

Esta última se encarga de llamar a la función que selecciona cual es la próxima acción a ejecutar (*syscallHandler*), y luego llama a la ejecución de dicha acción. Una vez llevada a cabo, da por finalizada la rutina de atención de la interrupción.

Loader (loader.asm) y Kernel (kernel.c)

Inicialmente, se cargan en memoria los módulos del Kernel Land y del User Land.
Inmediatamente después, se hace una limpieza de la BSS del Kernel Land, para poder tener datos consistentes a lo largo de la ejecución del sistema.

A continuación, se deshabilitan las interrupciones de hardware para inicializar la IDT (Interrupt Descriptor Table). Los drivers de teclado y timer tick son guardados en la IDT como los servicios de atención de interrupción de los dispositivos que llevan su nombre.
(Nota: la IDT es creada por el BootLoader provisto por la cátedra; se utilizó esa misma IDT, que se encuentra en la posición 0 de memoria).

Además, se guarda en la IDT la rutina de atención de la interrupción int 80h, que será usada como mediadora entre el Kernel Land y el User Land.

Una vez finalizado el guardado de rutinas en la IDT, se procede a habilitar sólo las interrupciones de hardware para las cuales se establecieron los respectivos servicios de atención de interrupción. Para lograr esto, se utiliza tanto la máscara del PIC como el IF flag del procesador (habilitado mediante la instrucción sti de assembly).

Finalmente, se llama al main del archivo *shell.c*, para comenzar la interacción con el usuario a través de un modelo de terminal.

User Land

Abarca los diferentes módulos (aparte del de kernel) que se cargan en el sistema.

En esta implementación se cuenta con un único módulo Shell que contiene sus archivos fuente programados en C ubicados en *src*, sus archivos programados en assembly en la carpeta *asm* y sus includes en la carpeta *include* mientras que el archivo principal del módulo se encuentra en la carpeta principal con el nombre *shell.c*.

El archivo *define.h* (ubicado en *include*) contiene definiciones generales que se utilizan a lo largo de todo el módulo.
Este módulo contiene 3 librerías estándares que facilitan funcionalidades que requieren interacción con el Kernel. Estas son:

- *libc.c*: contiene funciones de entrada y salida; manejo de strings y números; manejo de memoria. Ej: putchar, scanf, printf, memset, etc.
- *screen.c*: contiene las funcionalidades necesarias para el manejo del screen. Esto es: borrar la pantalla, interactuar con el screensaver, cambiar de terminal, etc.
- *time.c*: contiene todas las funciones para el manejo del reloj del sistema así como utilidades de tiempo. Ej: validar fecha.

Syscalls

Las librerías mencionadas utilizan las funciones primitivas del archivo *primitives.c* (ubicado en *src*) que representan cada una de las syscalls disponibles en el sistema. Estas son:

read (0)	write (1)	screen clear (2)
set terminal (3)	terminal colors (4)	time read (5)
time write (6)	enable screensaver (7)	set screensaver time (8)

Las primitivas se comunican con el Kernel mediante la syscall ubicada en el archivo *int80.asm* que contiene una llamada genérica a la int 80h. Por esta razón, en definitiva, las primitivas sólo se encargan de garantizar la correcta invocación de la int 80h, completando los campos de parámetros que no fueron entregados por la syscall antes mencionada.

Shell (shell.c)

La función main de este archivo hace las veces de un intérprete de comandos que le permite al usuario interactuar con el sistema. *Nota:* lo primero que se hace al ejecutar esta función es limpiar la BSS del módulo de User Land, para poder tener datos consistentes a lo largo de la ejecución del sistema.

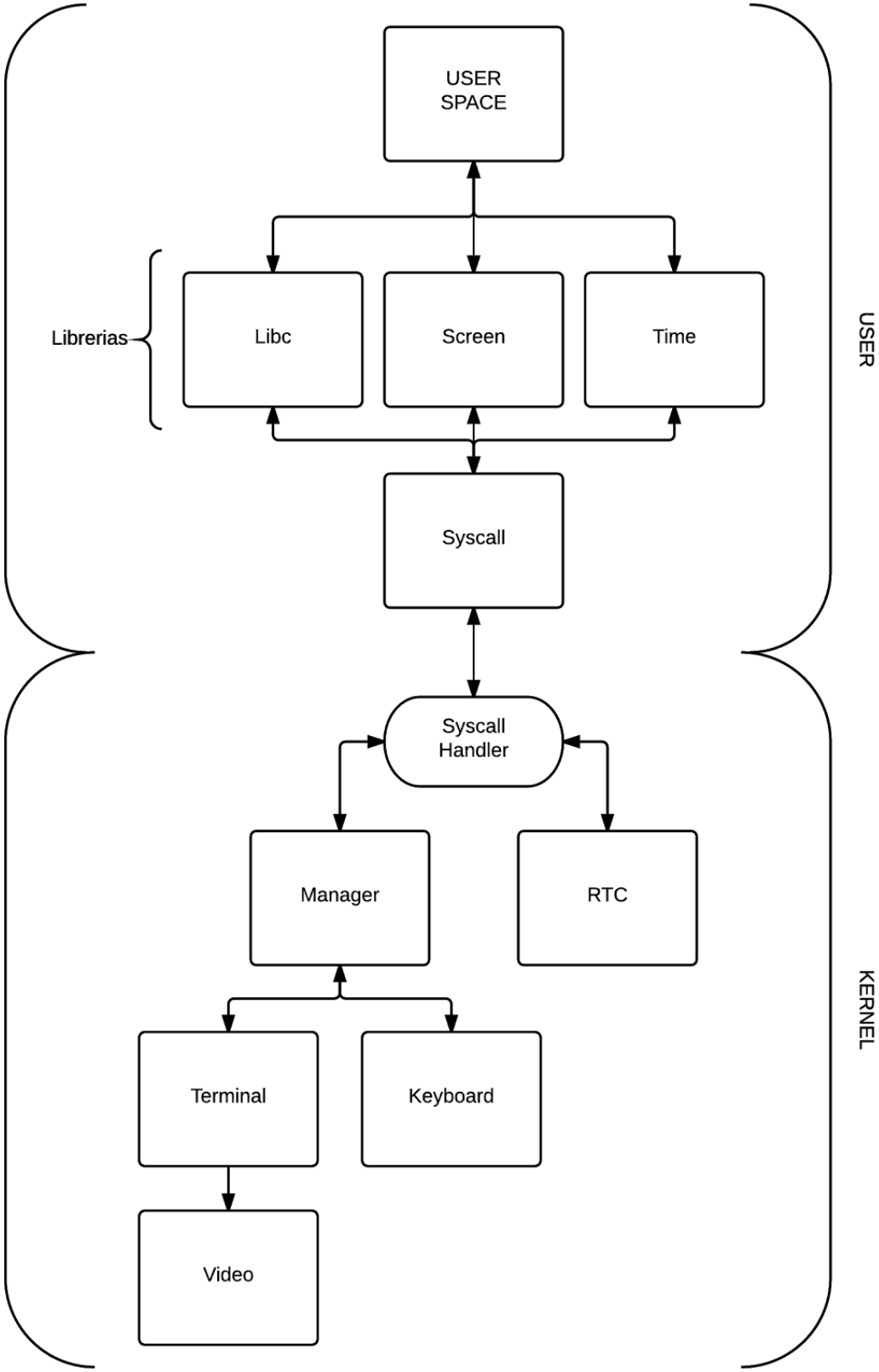
Los comandos que la shell interpreta están ubicados en el archivo *commands.c* (ubicado a su vez en *src*). Estos utilizan las librerías estándares para comunicarse con el sistema. La lista de comandos es la siguiente:

help	user*
clear*	terminal*
text*	background*
screensaver	time

* comandos adicionales a lo requerido

Adicionalmente, presionar la tecla TAB permite cambiar el layout del teclado (Inglés US o Español).
Para mayor detalle, ver *Manual de Usuario*.

Interacción entre Kernel Land y User Land



Decisiones de implementación

Implementación del Teclado

Al recibir una interrupción de teclado, la rutina de atención de interrupción correspondiente es una función que recibe el scan code (código) de la tecla presionada / soltada, y la mapea con un caracter o función, según el estado actual de los flags del teclado.
Estos flags son seteados según las teclas funcionales (entre ellas, shift, control, alt, bloq mayusc., bloq num, etc.) que fueron o están siendo apretadas al momento de procesar el scan code actual.
Una vez mapeada, la tecla es enviada al manager para que realice la operación correspondiente, tal como fue explicado anteriormente.

Internamente, el teclado tiene un buffer circular, que es usado por el manager para almacenar teclas. Una vez llenado este buffer, todos los caracteres que se quieran almacenar son ignorados, con excepción del próximo *enter* (que habilitará la lectura del buffer).
Nuestro modelo utiliza la siguiente implementación:
Las teclas mapeadas con funciones (a las que llamamos "*funcionales"*) son consumidas inmediatamente, ya sea porque el keyboard las utiliza para establecer sus propios flags, o porque el manager las utiliza para ejecutar diferentes acciones.
Debido a esto, solo fue necesario que las teclas de tipo *"imprimible"* sean almacenadas en el buffer, para luego poder ser consumidas (no directamente, sino a través del sistema de interrupciones correspondiente) por funciones como getchar.

La función que realiza el almacenamiento de una tecla se fija si el caracter asociado a esa tecla es un *enter*. De ser así, no sólo la almacena, sino que además, aumenta un contador de *enters* actuales en el buffer de teclado.

Este contador es el que permite que el buffer del teclado pueda ser aprovechado de manera óptima, por lo que sigue.
Mientras el contador de *enters* actuales esté en cero, ninguna función que quiera leer de entrada estándar podrá hacerlo. Para ello, deberá esperar a que el contador valga algún número mayor que cero. De esta forma, se brinda la posibilidad de realizar la cantidad de modificaciones que se desee sobre la línea a leer, puesto que los backspaces son interpretados directamente sobre el buffer al ser ingresados. Además, gracias a esto, las funciones que quieran leer el buffer no deberán recorrerlos dos veces (una limpiando los backspaces y otra leyendo el mensaje completo), sino que el mensaje ya estará listo para ser procesado.
Cuando el contador de *enters* actuales en el buffer de teclado es mayor que cero, se habilita la lectura del buffer para cualquier función que quiera leer caracteres de la entrada estándar.

Scanf implementado como un ciclo de getchar

Esta implementación de scanf fue desarrollada de esta forma para tener un mayor control sobre lo que se lee de la entrada estándar. Además, se trató de imitar la forma en que se programa un getLine o scanf propio de C, cuando no se conoce lo que en realidad hace internamente getchar.
Si bien la interrupción para leer desde la entrada estándar está preparada para tomar más caracteres que de a uno por vez, se decidió no utilizar dicha implementación por lo mencionado anteriormente, pero si tener preparada la estructura para en el caso de querer usarla en un futuro.

Implementación de Terminales

Se decidió implementar la posibilidad de poder disponer de varias terminales individuales con la posibilidad de volcar en memoria de video una de ellas (llamada terminal actual) que es donde va a correr el módulo. Para trabajar con estas terminales se implementó un comando “terminal set <num>” para poder cambiar entre ellas y se armó toda la funcionalidad para poder guardar cada terminal en memoria e ir volcando la actual en la memoria de video.

El sistema se hizo completamente expandible por lo que se pueden agregar terminales con facilidad (modificando código) y el mismo permite generar una copia de la terminal actual para poder restaurar luego, lo que facilita la implementación del screensaver que se detalla luego.

Lo que hace la implementación es guardar todo el estado del screen, esto incluye todos los caracteres, los estilos de los mismo, la posición del cursor y el estilo actual.

Al utilizar el comando “terminal set <num>” se procede a actualizar la copia ya generada del screen actual y se copia la terminal deseada a memoria de video en donde va a seguir corriendo la shell.

Una futura implementación puede permitir modificar varias terminales al mismo tiempo e ir volcando la que se desea ver en memoria de video y así permitir un multitasking pero escapa de los conocimientos otorgados por la materia.

Keyboard en Manager

Se procedió a implementar que el Manager sea el mediador del keyboard con el sistema (y no que Terminal lo sea) para poder independizar al teclado de cada una de las terminales y para poder hacer una abstracción entre cada terminal y el teclado (es decir, que la terminal no tenga referencia del teclado, sino que sólo se encargue de administrar el driver de video).

Screensaver

El screensaver consiste en actualizar la copia de la terminal actual en memoria y proceder a borrar el contenido de la memoria de video. Esto permite poder escribir lo que se quiera sobre la pantalla sin modificar el estado verdadero de la terminal.

El mismo se activa cuando el contador del manager, que se suma cuando el timer tick genera una notificación, llega a la cuenta deseada y se procede a copiar el screen, borrar la pantalla y empezar a escribir sobre pantalla lo que se desea. Para apagarlo se debe presionar una tecla cualquiera que no va a quedar presente en el buffer del teclado para que no se lea luego. Para esto se bloquea la escritura sobre buffer mientras el screensaver este corriendo. Al detectarse la tecla se restaura el screen que estaba como actual antes de encenderse el screensaver y se habilita la escritura sobre el buffer.

Funcionalidades añadidas al Screensaver

Además de poder configurar el tiempo que debe pasar para que se corra el mismo, se implementó la funcionalidad de habilitarlo y deshabilitarlo.

Problemas con RTC

Al correr el sistema utilizando qemu en computadoras personales, surgieron dos problemas:

1- No se podían configurar años mayores que el 2300 ni menores que el 1770.

2- No se podía elegir configurar el día igual a 31 para los meses válidos (los que realmente tienen 31 días).

Mientras que al correrlo utilizando Virtual Box surgió el siguiente problema: al querer pasar del año 1999 al 2000, se produce un error, y se configura el año como 1900.

Se decidió que las funciones de *rtc.c* estuviesen preparadas para lidiar con el error de actualización de “siglo” de Virtual Box, por lo que si se trata de correr el sistema con qemu y hacer un cambio de siglo, ahora se producirá un error.

Además, se decidió acotar los años válidos entre 1980 y 2250, pues eran parámetros válidos para la versión de qemu sobre la que se mencionaron los anteriores problemas.

Nota: Con la versión de qemu de los laboratorios de informática del I.T.B.A. (QEMU emulator version 1.0), la actualización del “siglo” sufre el mismo problema que sufría en Virtual Box (la mencionada anteriormente). Por ello, la solución adoptada por el grupo soluciona también el problema en este caso.

Funcionalidades añadidas con RTC

Además de poder mostrar y configurar la hora del sistema, se permite mostrar y configurar la fecha del mismo.

Códigos modificados de terceros

- Lectura del RTC: http://wiki.osdev.org/CMOS#Writing_to_the_CMOS
- Uso del cursor: http://wiki.osdev.org/Text_Mode_Cursor