DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP III - System Programming: TronTank

Grupo: Alemania / Vollkornbrot

Organización del Computador II
Primer Cuatrimestre de 2014

Integrante	LU	Correo electrónico
Quiroz, Nicolás	450/11	nh.quiroz@gmail.com
Rodríguez, Pedro	197/12	pedrorodriguezsjs@hotmail.com
Vuotto, Lucas	385/12	lvuotto@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

Una computadora, al iniciar, comienza con la ejecución del POST y luego el BIOS, el cual se encarga de reconocer el primer dispositivo de booteo. En lo que concierne a este trabajo práctico, disponemos de un floppy disk como unidad booteable. En el primer sector de dicha unidad se encuentra el *boot-sector*. El BIOS se encarga de copiar a memoria 512 bytes de este sector, a partir de la dirección `0x7c00`. Luego, se comienza a ejecutar el código a partir de esta dirección. El boot-sector debe encontrar en el *floppy* el archivo `kernel.bin` y copiarlo a memoria. Éste se copia a partir de la direción `0x1200`, y luego se ejecuta a partir de esa misma dirección. En la figura 1 se presenta el mapa de organización de la memoria utilizada por el *kernel*. Es importante tener en cuenta que el código del *boot-sector* se encarga exclusivamente de copiar el *kernel* y dar el control al mismo, es decir, no cambia el modo del procesador. El código del *boot-sector*, como así todo el esquema de trabajo para armar el *kernel* y ejecutar tareas, es provisto por la cátedra.

Índice

1. Objetivos generales

El objetivo de este trabajo práctico consiste en realizar una serie de ejercicios en los que aplicamos, gradualmente, los conceptos de programación de sistemas, que vimos durante la segunda parte de la cursada. El sistema desarrollado debe ser capaz de capturar cualquier problema generado por las tareas (exactamente 8, a nivel de usuario) y tomar las medidas necesarias para neutralizar las mismas y quitarlas, utilizando los mecanismos propios del procesador, desde el punto de vista del sistema operativo, enfocándonos esencialmente en dos aspectos: el sistema de protección y la ejecución concurrente de tareas.

2. Plataforma de pruebas

Como entorno de pruebas, utilizamos el software ***Bochs***. Este programa, de código abierto, nos permite emular una *IBM-PC*, con arquitectura *Intel x86*, dispositivos comunes de E/S, y un BIOS. También puede ser compilado para emular 386, 486, Pentium/Pentium II/Pentium III/Pentium 4 o una CPU con arquitectura x86-64, incluyendo instrucciones adicionales, como las MMX, SSEx y 3DNow!. Es capaz de ejecutar la mayoría de los sistemas operativos, incluyendo Linux, DOS o Windows NT/2000/XP/Vista/Seven.
Su uso típico es el de proveer una emulación completa de una PC x86, incluyendo al procesador y el resto del hardware y periféricos (memoria, discos duros, teclado, unidad de cdrom, disquetes, etc.), pero también es muy utilizado para la depuración de sistemas (*debugging*), ya que, si el sistema operativo huésped cae, por alguna razón, el sistema operativo anfitrión no cae también. Además, lleva un registro de errores y volcado de archivos. De esta forma, tenemos üna máquina dentro de la máquina".

Fuente: *Bochs User Manual,* http://bochs.sourceforge.net/doc/docbook/user/index.html

El desarrollo de este trabajo práctico fue realizado, principalmente, en la **máquina 17 del laboratorio 5 del DC**.

- **Sistema Operativo:** Ubuntu Linux 12.04 x86_64, kernel 3.2.0-30-generic
- **Especificaciones del Software:** Bochs IA-32 Emulator Project, versión 2.6.2. [1]
- **Especificaciones del Hardware:** Intel ® Core(TM) 2 Quad CPU Q9650 @3.00 Ghz, 4GB de RAM, memoria caché de 6144 KB.

[1] **Descarga y changelog:** http://sourceforge.net/projects/bochs/files/bochs/2.6.2/

3. Enunciado y solución

Ejercicio 1

a) Completar la Tabla de Descriptores Globales (`GDT`) con 4 segmentos, dos para código de nivel 0 y 3; y otros dos para datos de nivel 0 y 3. Estos segmentos deben direccionar los primeros 733MB de memoria. En la *gdt*, por restricción del trabajo práctico, las primeras 8 posiciones se consideran utilizadas y no deben utilizarse. El primer índice que deben usar para declarar los segmentos, es el 9 (contando desde cero).

completar.

b) Completar el código necesario para pasar a modo protegido y setear la pila del *kernel* en la dirección `0x27000`.

completar.

c) Declarar un segmento adicional que describa el área de la pantalla en memoria que pueda ser utilizado sólo por el *kernel*.

completar.

d) Escribir una rutina que se encargue de limpiar la pantalla y pintar en el área de *el_mapa* un fondo de color (sugerido verde). Para este ejercicio se debe escribir en la pantalla usando el segmento declarado en el punto anterior (para los próximos ejercicios se accederá a la memoria de video por medio del segmento de datos de 733MB).

completar.

Ejercicio 2

a) Completar las entradas necesarias en la `IDT` para asociar diferentes rutinas a todas las excepciones del procesador. Cada rutina de excepción debe indicar en pantalla qué problema se produjo e interrumpir la ejecución. Posteriormente se modificarán estas rutinas para que se continúe la ejecución, resolviendo el problema y desalojando a la tarea que lo produjo.

completar.

b) Hacer lo necesario para que el procesador utilice la `IDT` creada anteriormente. Generar una excepción para probarla.

completar.

Ejercicio 3

a) Escribir una rutina que se encargue de limpiar el *buffer* de video y pintarlo como indica la figura 8. Tener en cuenta que deben ser escritos de forma genérica para, posteriormente, ser completados con información del sistema. Además, considerar estas imágenes como sugerencias, ya que pueden ser modificadas a gusto según cada grupo, mostrando siempre la misma información.

completar.

b) Escribir las rutinas encargadas de inicializar el directorio y tablas de páginas para el *kernel* (`mmu_inicializar_dir_kernel`). Se debe generar un directorio de páginas que mapee, usando *identity mapping*, las direcciones `0x00000000` a `0x00DC3FFF`, como ilustra la figura 5. Además, esta función debe inicializar el directorio de páginas en la dirección `0x27000` y las tablas de páginas según muestra la figura 1.

completar.

c) Completar el código necesario para activar paginación.

completar.

d) Escribir una rutina que imprima el nombre del grupo en pantalla. Debe estar ubicado en la primer línea de la pantalla, alineado a la derecha.

completar.

Ejercicio 4

a) Escribir una rutina (`inicializar_mmu`), que se encargue de inicializar las estructuras necesarias para administrar la memoria en el área libre.

completar.

b) Escribir una rutina (`mmu_inicializar_dir_tarea`), encargada de inicializar un directorio de páginas y tablas de páginas para una tarea, respetando la figura 5 (ver enunciado). La rutina debe copiar el código de la tarea a su área asignada, es decir, sus dos páginas de código dentro de *el_mapa* y mapear dichas páginas a partir de la dirección virtual (`0x08000000`) (128MB).

completar.

c) Escribir dos rutinas encargadas de mapear y desmapear páginas de memoria.

1. `mmu_mapear_pagina(unsigned int virtual, unsigned int cr3, unsigned int fisica)` Permite mapear la página física correspondiente a `fisica` en la dirección virtual `virtual` utilizando `cr3`.
2. `mmu_unmapear_pagina(unsigned int virtual, unsigned int cr3)` Borra el mapeo creado en la dirección virtual `virtual` utilizando `cr3`.

completar.

d) Construir un mapa de memoria para tareas e intercambiarlo con el del *kernel*, luego cambiar el color del fondo del primer caracter de la pantalla y volver a la normalidad.

completar.

Ejercicio 5

completar.

Ejercicio 6

completar.

Ejercicio 7

completar.

Ejercicio 8 (optativo)

completar.

4. Conclusiones

completar.