DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

TronTank

Organización del Computador II

Primer Cuatrimestre de 2014

Integrante	LU	Correo electrónico
Tomas Shaurli	671/10	tshaurli@gmail.com
Sebastian Aversa	379/11	sebastianaversa@gmail.com
Fernando Gabriel Otero	424/11	fergabot@gmail.com

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

En el presente trabajo se realiza la configuración de un kernel de sistema operativo de 32bits con sus elementos indispensables (GDT, IDT, TSS, paginación, etc) y el manejo de tareas.

Índice

1. Objetivos generales

Este trabajo práctico consistió en un conjunto de ejercicios para el parendizaje de los conceptos de System Programming
Se implementó para ello un sistema mínimo en base a los archivos aportados por la cátedra, el cual será capaz de manejar exactamente 8 tareas a nivel de usuario. El sistema será capaz de capturar cualquier problema que puedan generar las tareas y tomar las acciones necesarias para quitar a la tarea del sistema.
Los ejercicios de este trabajo práctico proponen utilizar los mecanismos que posee el procesador para la programación desde el punto de vista del sistema operativo enfocados en dos aspectos: el sistema de protección y la ejecución concurrente de tareas.

2. Kernel.asm

2.1. Básico:

En el kernel lo primero que hacemos es deshabilitar las interrupciones con **CLI** y cambiamos el modo de video. Luego, iremos habilitando y creando los diversos componentes, llamando a las funciones destinadas para ése fin.

Esas funciones son listadas al comienzo del código:

```
extern GDT_DESC
extern IDT_DESC
extern idt_inicializar
extern mmu_inicializar
extern mmu_inicializar_dir_kernel
extern pintar
extern pintarTablero
extern deshabilitar_pic
extern resetear_pic
extern habilitar_pic
```

2.2. Preparando el modo protegido:

Lo segundo que tenemos que hacer es habilitar A20 en el controlador del teclado para tener acceso a direcciones superiores a los 2^{20} bits

Luego, cargamos la **GDT** con *LGDT*, pasandole el descriptor de la GDT que ya tiene el formato adecuado

```
    ; Habilitar A20
    CALL habilitar_A20
    ; Cargar la GDT
    LGDT [GDT_DESC]
```

Ahora, con la GDT cargada y A20 habilitado, podemos pasar a modo protegido seteando el bit PE de CR0 (bit 0) en 1.

Al finalizar debemos hacer un *far jump* a la etiqueta de modo protegido. Es importante notar que el far jump es la unica forma de modificar el CS, operación necesaria para que no haya errores.

```
; Setear el bit PE del registro CR0 (esto pasa a modo protegido)
    MOV EAX,CR0
    OR EAX,1
    MOV CR0,EAX
    ; Saltar a modo protegido
    JMP (9*0x08):modoProtegido
```

2.3. Modo protegido:

Ya estamos en modo protegido. Ahora seteamos los selectores de segmentos haciendo uso de la GDT.

```
BITS 32
modoProtegido:
    ;CODIGO
; Establecer selectores de segmentos
XOR EAX, EAX
MOV AX, 1011000b ;1011b == 11d (index de la GDT) | 0 (0 -> GDT / 1 -> LDT) |
                                                  | 00 (NIVEL DE PRIVILEGIO)
MOV DS, AX
MOV ES, AX
MOV GS, AX
MOV SS, AX
MOV AX, 1101000b
MOV FS, AX
```

Establecemos la base de la pila, llamamos a *pintarTablero()* que limpia la pantalla y la pinta como en la figura 8 del enunciado. Luego, llamamos a idt_inicializar y, una vez inicializada, le pasamos el descriptor de la **IDT** a *LIDT*

Por ultimo pintamos

```
; Establecer la base de la pila
MOV ESP, 0x27000
MOV EBP, ESP
CALL pintarTablero
CALL idt_inicializar
LIDT [IDT_DESC]
```

2.4. Paginación:

Primero inicializamos las páginas con *identity mapping* y cargo en CR3 la direccion del *page directory* Luego modifico CR0 para habilitar paginación

```
CALL mmu_inicializar_dir_kernel
MOV EAX, 0x27000
MOV CR3, EAX ;cargo en CR3 la direccion del page directory
MOV EAX, CR0
    OR EAX, 0x80000000 ;habilitamos paginacion
MOV CR0, EAX
```

Algo

```
; Imprimir mensaje de bienvenida

; Inicializar pantalla

; Inicializar el manejador de memoria
```

Algo directorio de paginas de tanques?

```
; Inicializar el directorio de paginas
CALL mmu_inicializar

CALL deshabilitar_pic
CALL resetear_pic
CALL habilitar_pic
STI
```

RESTO

3. GDT

3.1. gdt.h

En el archivo gdt.h sólo se modifico el define GDT_COUNT, dándole 20 por valor ya que es la cantidad de entradas que tiene la GDT

3.2. gdt.c

En gdt.c se tomó la extructura dada por la cátedra y se agregaron las entradas correspondientes.

- La entrada nula, en la posicion 0
- Dos descriptores de segmentos de datos, uno de prioridad 0 y otro de prioridad 3
- Dos descriptores de segmentos de código, uno de prioridad 0 y otro de prioridad 3
- El descriptor del segmento de video
- Tres descriptores de TSS, el primero para la tarea *Idle*

En el archivo defines.h esta definido el índice que tiene cada entrada en la tabla.

```
#define GDT_IDX_NULL_DESC           0
#define GDT_IDX_CODE_DESC_1 9
#define GDT_IDX_CODE_DESC_2 10
#define GDT_IDX_DATA_DESC_1 11
#define GDT_IDX_DATA_DESC_2 12
#define GDT_IDX_VIDEO_DESC_1 13
#define GDT_IDX_TSS_INICIAL 14
#define GDT_IDX_TSS1 15
#define GDT_IDX_TSS2 16
```

4. Video

4.1. screen.h:

En screen.h definimos las funciones que hacen uso de la pantalla, y definimos mem_video pasandole como valor la posicion de memoria donde comienza el segmento de video.

También definimos el struct str_memoria_video que representa los pixeles con el formato para ser mostrados por pantalla.

```
/* Definicion de la pantalla */
#define VIDEO_FILS 50
#define VIDEO_COLS 80
#define mem_vid 0xb8000

typedef struct str_memoria_video {
    unsigned char   ascii:8;
    unsigned char   caracter:3;
    unsigned char   brillante:1;
    unsigned char   fondo:3;
    unsigned char   blink:1;
} __attribute__((__packed__)) memoria_video;

void pintar();
void pintarTablero();
void mostrar_int(int i);
void mostrar_num(int n);
```

4.2. screen.c:

4.2.1. void pintarTablero()

Pinta la pantalla como en la figura 8 del enunciado.

```
void pintarTablero(){

    memoria_video* vd = (memoria_video*) (0xb8000);

    int f = 0;
    int c;
    while(f < VIDEO_FILS ){
        c = 0;
        while (c < VIDEO_COLS){
            if (c <= 50)
                vd->fondo = C_FG_GREEN;
            if (c >= 52)
            {
                if(f == 0 || f == 39)
                    vd->fondo = C_FG_RED;
                if( 0 < f && f < 39)
                    vd->fondo = C_FG_LIGHT_GREY;
            }
            if ( (52 < c && c < 70) && f >= 47)
                vd->fondo = C_FG_LIGHT_GREY;

            vd->ascii = (unsigned char) 0x0;
            vd++;
            c++;
        }
        f++;
    }
}
```

4.2.2. void imprimir(char* m, memoria_video* vd)

Recibe una cadena de caracteres y un formato, y la imprime por pantalla.

```
void imprimir(char* m, memoria_video* vd){

    memoria_video* vdAux = vd;
    while (*m != '\0'){
        vdAux->ascii = *m;
        m++;
        vdAux++;
    }
}
```

4.2.3. void mostrar_int(int teclado)

Esta función recivirá un entero equivalente al valor de una interrupción, entre 0 y 19

4.2.4. void mostrar_num(int teclado)

Avanza 79 posiciones luego del inicio del segmento de video, y escribe un entero determinado por pantalla. Solo se usa con los enteros 2, 3 y 4

```
void mostrar_num(int teclado){
    memoria_video* vd = (memoria_video*) (0xb8000);
    vd += 79;
    switch ( teclado)
    {
        case 2: imprimir("1", vd);
                break;
        case 3: imprimir("2", vd);
                break;
        case 4: imprimir("3", vd);
                break;
    }
}
```

5. Interrupciones de sistema

5.1. idt.c

La idt se genera utilizando la macro dada por la catedra (completada por nosotros).

```
#define IDT_ENTRY(numero)
    idt[numero].offset_0_15 = (unsigned short) ((unsigned int)(&_isr ## numero) & (unsigned int) 0xFFFF);
    idt[numero].segsel = (unsigned short) (GDT_IDX_CODE_DESC_1*8);
    idt[numero].attr = (unsigned short) 0x8E00;
    idt[numero].offset_16_31 = (unsigned short) ((unsigned int)(&_isr ## numero) >> 16 & (unsigned int) 0xFFFF);
```

Esta macro se usa dada la similitud entre todas las interrupciones manejadas en este archivo. De hecho, en idt.c la unica diferencia entre dos interrupciones es su número

5.2. screen.c

Aquí se encuentra la función *mostrar_int(int error)* con los mensajes correspondentes para cada interrupcion

5.3. isr.asm

En el archivo isr.asm completamos la información de las rutinas de atencion de interrupciones

6. MMU

6.1. A modificar

6.1.1. codigo util

cli *cli*

```
    ; Habilitar A20
    CALL habilitar_A20
    ; Cargar la GDT
    LGDT [GDT_DESC]
```

7. Interrupciones de reloj y teclado

```
;;
;; Rutina de atención del RELOJ
;; -------------------------------------------------------------------------- ;;
_isr32:
    CLI
    PUSHA
    CALL fin_intr_pic1
    ; CODIGO DE LA INTR

    CALL proximo_reloj

    POPA
    STI
    IRET

;;
;; Rutina de atención del TECLADO
;; -------------------------------------------------------------------------- ;;
_isr33:
CLI
PUSHA
XOR EAX, EAX
IN AL, 0x60
;xchg bx, bx
PUSH EAX
CALL mostrar_num
POP EAX
POPA
STI
IRET
```

8. TSS

8.1. A modificar

8.1.1. codigo util

cli *cli*

```
    ; Habilitar A20
    CALL habilitar_A20
    ; Cargar la GDT
    LGDT [GDT_DESC]
```

9. Scheduler

9.1. A modificar

9.1.1. codigo util

cli *cli*

```
    ; Habilitar A20
    CALL habilitar_A20
    ; Cargar la GDT
    LGDT [GDT_DESC]
```