Arquitectura de las Computadoras

TP Especial 2011

Modo protegido con GRUB

31 de mayo de 2011

Autores:

Axel Wassington	Legajo: *50124*
Horacio Miguel Gomez	Legajo: *50825*
Tomás Mehdi	Legajo: *51014*

Índice

1. Abstract

El TP Especial consta de un sistema booteable con GRUB instalado, el cual debe obtener de disco una imagen binaria que correra en Modo protegido. El mismo provee al usuario de un shell en el cual puede ejecutar comandos. Similar a Unix/Linux utilizamos la INT80h, mediante la cual se pueden ejecutar las instrucciones de assembler IN, OUT y otras. Al presionar CTRL+ALT+SUPR el sistema se reinicia.

2. Manual de uso

Esta seccion esta destinada a proporcionar información sobre las funcionalidades del sistema y explicar su modo de uso.

2.1. Booteo

El sistema es booteable, por lo que para su inicio solamente se requiere que el CD se encuentre insertado y la lectora posea la mayor prioridad de booteo. Luego se mostrará la pantalla de GRUB. Se puede presionar enter para proceder, o de lo contrario GRUB continuar á la carga del sistema tras un número prestablecido de segundos, visible en el ultimo renglón de la pantalla. Una vez realizado el booteo, se cargará el shell. A partir de aquí se puede comenzar a utilizar los comandos provistos por el shell.

2.2. Comandos de shell

La siguiente lista indica la forma de usar a los distintos comandos de el shell. Es importante destacar que todos las llamadas son case sensitive. En caso de ingresar un comando que no esta contemplado por el shell se imprime un mensaje de error y se devuelve el control al usuario.

2.2.1. cpuspeed

Muestra la frecuencia de trabajo del CPU, usando la funcion RDTSC(Read Time-Stamp Counter) de assembler la cual retorna la cantidad de instrucciones realizada hasta el momento desde el inicio del procesador y el PIT(Programable Interval Timer). El PIT es un periferico conectado a la IRQ0 del the master PIC. Utilizando estos elementos podemos obtener una cantidad de instrucciones en un intervalo de tiempo. La forma de hacerlo es obteniendo un RDTSC, dejar pasar un tiempo fijo y volver a obtener un RDTSC. El tiempo fijo lo generamos con una cantidad coherente de timer ticks, para ello no debe ser muy pequeña. Ya teniendo estos datos solo falta hacer simples cuentas que nos devolveran la velocidad del CPU en MHz.

2.2.2. clear

Limpia la pantalla.

2.2.3. echo

Recice una cadena de caracteres luego del nombre del comando e imprime dicha cadena.

2.2.4. exit

Cierra el shell.

3. INT80h

Similar a la INT80h de Unix/Linux la INT80h de Arnix segun el valor en el registro EAX elige una instruccion. Las instrucciones que puede realizar son las siguientes:

1. Con el valor 3 en EAX hace un read usando los valores de EBX, ECX y EDX. En estos regristros debe estar el tamaño de lo que se va a leer, el source buffer y un file descriptor(de donde leer) respectivamente.

2. Con el valor 4 en EAX hace un write usando los valores de EBX, ECX y EDX. En estos regristros debe estar el tamaño de lo que se va a escribir, el source buffer y un file descriptor(donde esribir) respectivamente.

3. Con el valor 5 en EAX hace un rdtsc guardando el valor en el registro EBX.

4. Codigos fuente

4.1. include

defs.h

```
/***************************************************
  Defs.h

***************************************************/

#ifndef _defs_
#define _defs_

#define byte unsigned char
#define word short int
#define dword int

/* Flags para derechos de acceso de los segmentos */
#define ACS_PRESENT     0x80            /* segmento presente en ↩
    memoria */
#define ACS_CSEG        0x18            /* segmento de codigo */
#define ACS_DSEG        0x10            /* segmento de datos */
#define ACS_READ        0x02            /* segmento de lectura */
#define ACS_WRITE       0x02            /* segmento de escritura */
#define ACS_IDT         ACS_DSEG
#define ACS_INT_386     0x0E        /* Interrupt GATE 32 bits */
#define ACS_INT         ( ACS_PRESENT | ACS_INT_386 )


#define ACS_CODE        (ACS_PRESENT | ACS_CSEG | ACS_READ)
#define ACS_DATA        (ACS_PRESENT | ACS_DSEG | ACS_WRITE)
#define ACS_STACK       (ACS_PRESENT | ACS_DSEG | ACS_WRITE)

#pragma pack (1)        /* Alinear las siguiente estructuras a 1 byte ↩
    */

/* Descriptor de segmento */
typedef struct {
  word limit,
       base_l;
  byte base_m,
       access,
       attribs,
       base_h;
} DESCR_SEG;


/* Descriptor de interrupcion */
typedef struct {
  word      offset_l,
            selector;
  byte      cero,
            access;
  word      offset_h;
} DESCR_INT;

/* IDTR  */
typedef struct {
  word  limit;
  dword base;
} IDTR;



#endif
```

kasm.h

```
/*********************************************
kasm.h

**********************************************/
#include "defs.h"


unsigned int    _read_msw();

void            _lidt (IDTR *idtr);

void        _mascaraPIC1 (byte mascara);  /* Escribe mascara de PIC1 */
void        _mascaraPIC2 (byte mascara);  /* Escribe mascara de PIC2 */

void        _Cli(void);        /* Deshabilita interrupciones  */
void        _Sti(void);  /* Habilita interrupciones */

void        _int_08_hand();      /* Timer tick */

void        _debug (void);
```

kc.h

```
/**********************
 kc.h
**********************/
#include "defs.h"

#ifndef _kc_
#define _kc_

#define WHITE_TXT 0x07 // Atributo de video. Letras blancas, fondo negro

/* Muestra la imagen de inicio */
void showSplashScreen();

/* Tiempo de espera */
void wait(int time);

/* Limpia la pantalla */
void k_clear_screen();

/* Inicializa la entrada del IDT */
void setup_IDT_entry (DESCR_INT *item, byte selector, dword offset, byte access,
             byte cero);

#endif
```

kernel.h

```
#include "../include/defs.h"

/********************************
*
*  Kernel
*
*********************************/

//#ifndef _kernel_
//#define _kernel_
//
//#define OS_PID     0
//
```

```
//int (*player)(void);
//
//typedef int size_t;
//typedef short int ssize_t;
//typedef enum eINT_80 {WRITE=0, READ} tINT_80;
//typedef enum eUSER {U_KERNEL=0, U_NORMAL} tUSERS;

/* __write
*
* Recibe como parametros:
* - File Descriptor
* - Buffer del source
* - Cantidad
*
**/


/* __read
*
* Recibe como parametros:
* - File Descriptor
* - Buffer a donde escribir
* - Cantidad
*
**/


#endif
```

stdarg.h

```
/*
 * stdarg.h
 *
 * Provides facilities for stepping through a list of function ↩
    arguments of
 * an unknown number and type.
 *
 * NOTE: Gcc should provide stdarg.h, and I believe their version will↩
     work
 *       with crtdll. If necessary I think you can replace this with ↩
    the GCC
 *       stdarg.h (or is it vararg.h).
 *
 * Note that the type used in va_arg is supposed to match the actual ↩
    type
 * *after default promotions*. Thus, va_arg (..., short) is not valid.
 *
 * This file is part of the Mingw32 package.
 *
 * Contributors:
 *  Created by Colin Peters <colin@bird.fu.is.saga-u.ac.jp>
 *
 *  THIS SOFTWARE IS NOT COPYRIGHTED
 *
 *  This source code is offered for use in the public domain. You may
 *  use, modify or distribute it freely.
 *
 *  This code is distributed in the hope that it will be useful but
 *  WITHOUT ANY WARRANTY. ALL WARRANTIES, EXPRESS OR IMPLIED ARE ↩
    HEREBY
 *  DISCLAMED. This includes but is not limited to warranties of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * $Revision: 1.1.1.1 $
 * $Author: brandon6684 $
 * $Date: 2001/12/18 22:53:51 $
 *
 */
/* Appropriated for Reactos Crtdll by Ariadne */

#ifndef STDARG_H
```

```
#define STDARG_H

/*
 * Don't do any of this stuff for the resource compiler.
 */
#ifndef RC_INVOKED

/*
 * I was told that Win NT likes this.
 */
#ifndef _VA_LIST_DEFINED
#define _VA_LIST_DEFINED
#endif

#ifndef _VA_LIST
#define _VA_LIST
typedef char* va_list;
#endif


/*
 * Amount of space required in an argument list (ie. the stack) for an
 * argument of type t.
 */
#define __va_argsiz(t)  \
    (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))


/*
 * Start variable argument list processing by setting AP to point to the
 * argument after pN.
 */
#ifdef  __GNUC__
/*
 * In GNU the stack is not necessarily arranged very neatly in order to
 * pack shorts and such into a smaller argument list. Fortunately a
 * neatly arranged version is available through the use of __builtin_next_arg.
 */
#define va_start(ap, pN)    \
    ((ap) = ((va_list) __builtin_next_arg(pN)))
#else
/*
 * For a simple minded compiler this should work (it works in GNU too for
 * vararg lists that don't follow shorts and such).
 */
#define va_start(ap, pN)    \
    ((ap) = ((va_list) (&pN) + __va_argsiz(pN)))
#endif


/*
 * End processing of variable argument list. In this case we do nothing.
 */
#define va_end(ap)  ((void)0)


/*
 * Increment ap to the next argument in the list while returing a
 * pointer to what ap pointed to first, which is of type t.
 *
 * We cast to void* and then to t* because this avoids a warning about
 * increasing the alignment requirement.
 */

#define va_arg(ap, t)                   \
     (((ap) = (ap) + __va_argsiz(t)),       \
      *((t*) (void*) ((ap) - __va_argsiz(t))))

#endif /* Not RC_INVOKED */
```

```
#endif /* not _STDARG_H_ */
```

varargs.h

```
/* $NetBSD: varargs.h,v 1.11 2005/12/11 12:16:16 christos Exp $ */

/*-
 * Copyright (c) 1990, 1993
 *  The Regents of the University of California.  All rights reserved.
 * (c) UNIX System Laboratories, Inc.
 * All or some portions of this file are derived from material ↩
    licensed
 * to the University of California by American Telephone and Telegraph
 * Co. or Unix System Laboratories, Inc. and are reproduced herein ↩
    with
 * the permission of UNIX System Laboratories, Inc.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above ↩
    copyright
 *    notice, this list of conditions and the following disclaimer in ↩
    the
 *    documentation and/or other materials provided with the ↩
    distribution.
 * 3. Neither the name of the University nor the names of its ↩
    contributors
 *    may be used to endorse or promote products derived from this ↩
    software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''↩
     AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, ↩
    THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR ↩
    PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE ↩
    LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR ↩
    CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE ↩
    GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS ↩
    INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,↩
     STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ↩
    ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY↩
     OF
 * SUCH DAMAGE.
 *
 *  @(#)varargs.h   8.2 (Berkeley) 3/22/94
 */

#ifndef VARARGS_H
#define VARARGS_H

#if !__GNUC_PREREQ__
#define __va_ellipsis
#else
#define __va_ellipsis   ...
#endif

#if __GNUC_PREREQ__
#define __va_alist_t    __builtin_va_alist_t
#else
```

```
#define __va_alist_t    long
#endif

#define va_alist      __builtin_va_alist
#define va_dcl        __va_alist_t __builtin_va_alist; __va_ellipsis


#endif
```

4.2. src

4.2.1. kernel

kernel.c

```
#include "../include/kasm.h"
#include "../include/defs.h"
#include "kernel/driver/screen.h"
#include "kernel/system/idt.h"
#include "kernel/driver/keyboard.h"
#include "kernel/system/keyboardlisteners.h"

DESCR_INT idt[0x80];            /* IDT de 80 entradas*/
IDTR idtr;              /* IDTR */

/**********************************************
kmain()
Punto de entrada de codigo C.
*************************************************/

kmain()
{
    init_descriptor_tables();
    init_int80();
    init_in_out();
    init_keyboard();
    init_timer_tick();
    init_screen();


    shell_start();
}
```

lib.c

```
#include "../include/kc.h"


/***************************************************************
*k_clear_screen
*
* Borra la pantalla en modo texto color.
****************************************************************/

void k_clear_screen()
{
    char *vidmem = (char *) 0xb8000;
    unsigned int i=0;
    while(i < (80*25*2))
    {
        vidmem[i]=' ';
        i++;
        vidmem[i]=WHITE_TXT;
        i++;
    };
}
```

```
/****************************************************************
*setup_IDT_entry
* Inicializa un descriptor de la IDT
*
*Recibe: Puntero a elemento de la IDT
*    Selector a cargar en el descriptor de interrupcion
*    Puntero a rutina de atencion de interrupcion
*    Derechos de acceso del segmento
*    Cero
****************************************************************/

void setup_IDT_entry (DESCR_INT *item, byte selector, dword offset, ↩
    byte access,
             byte cero) {
  item->selector = selector;
  item->offset_l = offset & 0xFFFF;
  item->offset_h = offset >> 16;
  item->access = access;
  item->cero = cero;
}
```

loader.asm

```
global _loader      ; making entry point visible to linker
global eokl     ; end of kernel land
extern kmain        ; _main is defined elsewhere


; setting up the Multiboot header - see GRUB docs for details
MODULEALIGN equ  1<<0                   ; align loaded modules on page↩
     boundaries
MEMINFO     equ  1<<1                   ; provide memory map
FLAGS       equ  MODULEALIGN | MEMINFO  ; this is the Multiboot 'flag'↩
     field
MAGIC       equ    0x1BADB002           ; 'magic number' lets ↩
    bootloader find the header
CHECKSUM    equ -(MAGIC + FLAGS)        ; checksum required

section .text
align 4
MultiBootHeader:
    dd MAGIC
    dd FLAGS
    dd CHECKSUM

    ; reserve initial kernel stack space
    STACKSIZE equ 0x4000        ; that's 16k.

    _loader:
    mov esp, stack+STACKSIZE; set up the stack
    push eax        ; pass Multiboot magic number
    push ebx        ; pass Multiboot info structure

    call  kmain     ; call kernel proper
    hlt         ; halt machine should kernel return

eokl    dd STACKSIZE + stack
    section .bss
    align 32
    stack:
    resb STACKSIZE      ; reserve 16k stack on a quadword boundary
```

4.2.2. driver

keyboard.h

```
#ifndef KEYBOARD_H
#define KEYBOARD_H

void init_keyboard();

#endif  /* KEYBOARD_H */
```

screen.c

```
#include "screen.h"
#include "../system/isr.h"
#include "../system/in_out.h"
#include "timer.h"

// The VGA framebuffer starts at 0xB8000.
int16_t *video_memory = (int16_t *)0xB8000;
// Stores the cursor position.

#define BUFFER_SIZE 1000

char array_out[BUFFER_SIZE];

buffer_t stdout;

#define ESC '\x1B'
#define BELL '\x07'

#define DEFAULT_SETTINGS 0x07

#define SCREEN_SIZE_X 80
#define SCREEN_SIZE_Y 25
uint8_t screen_state = 0; // 0=normal, 1=scaped, 2=parameters.

#define SCREEN_MAX_PARAM_COUNT 16
uint8_t screen_param_count = 0;
int screen_param[SCREEN_MAX_PARAM_COUNT];

uint8_t screen_cursor_x = 0;
uint8_t screen_cursor_y = 0;
uint8_t screen_settings = DEFAULT_SETTINGS;

static void update_cursor() {
    int16_t cursorLocation = screen_cursor_y * SCREEN_SIZE_X + ↩
        screen_cursor_x;
    outb(0x3D4, 14);                  // Tell the VGA board we are ↩
        setting the high cursor byte.
    outb(0x3D5, cursorLocation >> 8); // Send the high cursor byte.
    outb(0x3D4, 15);                  // Tell the VGA board we are ↩
        setting the low cursor byte.
    outb(0x3D5, cursorLocation);      // Send the low cursor byte.
}

// Scrolls the text on the screen up by one line.
static void scroll() {
    // Get a space character with the default colour attributes.
    uint8_t attributeByte = (0 /*black*/ << 4) | (15 /*white*/ & 0x0F)↩
        ;
    int16_t blank = 0x20 /* space */ | (attributeByte << 8);

    // Row SCREEN_SIZE_Y is the end, this means we need to scroll up
    if(screen_cursor_y >= SCREEN_SIZE_Y)
    {
        // Move the current text chunk that makes up the screen
        // back in the buffer by a line
        int i;
        for (i = 0*SCREEN_SIZE_X; i < (SCREEN_SIZE_Y-1)*SCREEN_SIZE_X;↩
             i++)
        {
            video_memory[i] = video_memory[i+SCREEN_SIZE_X];
        }
        int lastLine = SCREEN_SIZE_Y-1;
        // The last line should now be blank. Do this by writing
        // SCREEN_SIZE_X spaces to it.
        for (i = (lastLine)*SCREEN_SIZE_X; i < SCREEN_SIZE_Y*↩
            SCREEN_SIZE_X; i++)
        {
            video_memory[i] = blank;
        }
        screen_cursor_y = (lastLine);
    }
}

static void print(char c) {
    int16_t *location;
    location = video_memory + (screen_cursor_y*SCREEN_SIZE_X + ↩
        screen_cursor_x);

    if (c != '\b') {
        *location = (c | (screen_settings << 8));
        if (++screen_cursor_x >= SCREEN_SIZE_X) {
            screen_cursor_x = 0;
            screen_cursor_y ++;
        }
    } else {
        *location = (' ' | (screen_settings << 8));
    }
}

static void do_bell() {
   // TODO
}

static void do_backspace() {
    if(screen_cursor_x) {
        screen_cursor_x--;
    } else if (screen_cursor_y) {
        screen_cursor_x=SCREEN_SIZE_X-1;
        screen_cursor_y--;
    }
    print('\b');
}

static void do_lineFeed() {
    screen_cursor_x = 0;
    screen_cursor_y++;
}

static void do_tab() {
    screen_cursor_x = (screen_cursor_x+8) & ~(8-1);
}

static void do_return() {
    screen_cursor_x = 0;
}

// Clears the screen, by copying lots of spaces to the framebuffer.
static void screen_clear() {
    // Make an attribute byte for the default colours
    uint8_t attributeByte = (0 /*black*/ << 4) | (15 /*white*/ & 0x0F)↩
        ;
    int16_t blank = 0x20 /* space */ | (attributeByte << 8);

    int i;
    for (i = 0; i < SCREEN_SIZE_X*SCREEN_SIZE_Y; i++) {
        video_memory[i] = blank;
    }

    // Move the hardware cursor back to the start.
    screen_cursor_x = screen_cursor_y = 0;
    update_cursor();
}

static void do_scape_J() {
    if (screen_param[0] == 2) {
        screen_clear();
    }
}

/* Map from ANSI colors to the attributes used by the PC */
static uint8_t ansi_colors[8] = {0, 4, 2, 6, 1, 5, 3, 7};

static void do_scape_m() {
    int i;
    for (i=0;i<screen_param_count;i++){
        int dec = screen_param[i]/10;
        int u = screen_param[i]%10;
        if (dec == 0) {
            switch(u){
                case 0:
                    screen_settings = DEFAULT_SETTINGS;
                    break;
                case 1:
                    screen_settings |= 0x08;
                    break;
                case 4:
                    screen_settings &= 0xBB;
                    break;
                case 5:
                    screen_settings |= 0x80;
            }
        } else if (dec == 3) { /* foreground */
            //print('3');
            screen_settings = (0xF0 & screen_settings) | (0x0F & ↩
                ansi_colors[u]);
        } else if (dec == 4) { /* background */
            screen_settings = (0x0F & screen_settings) | (ansi_colors[↩
                u] << 4);
        }
    }
}

static void do_scape(char c) {
    switch(screen_state) {
        case 1:
            if (c == '[') {
                screen_state = 2;
                screen_param_count = 1;
                int i=0;
                for (;i<=SCREEN_MAX_PARAM_COUNT; i++) {
                    screen_param[i] = 0;
                }
            } else {
                screen_state = 0;
            }
            break;
        case 2:
            if (c >= '0' && c <= '9') {
                screen_param[screen_param_count-1] = 10*screen_param[↩
                    screen_param_count-1] + (c-'0');
            } else if (c == ';') {
                screen_param_count++;
            } else {
                switch (c) {
                    case 'm':
                        do_scape_m();
                        break;
                    case 'J':
                        do_scape_J();
                        break;
                }
                screen_state = 0;
            }
            break;
    }
}

// Writes a single character out to the screen.
void screen_put(char c) {
    if (screen_state > 0) {
        do_scape(c);
        return;
    } else {
        switch (c) {
            case ESC:
                screen_state = 1;
                return;
            case '\0':
                return;
            case BELL:
                do_bell();
                return;
            case '\b':
                do_backspace();
                break;
            case '\n':
                do_lineFeed();
                break;
            case '\t':
                do_tab();
                break;
            case '\r':
                do_return();
                break;
            default:
                print(c);
                break;
        }
        scroll();
        update_cursor();
    }
}

void screen_write(char *string) {
    int i = 0;
    while (string[i]) {
        screen_put(string[i++]);
    }
}

static void timer_print(registers_t reg){
    int i;
    for(i=0;stdout.start!=stdout.end;i++){
        screen_put(stdout.array[stdout.start]);
        stdout.start=(stdout.start+1)%stdout.size;
    }
}

void init_screen(){
    register_tick_subhandler(timer_print);
    stdout.start=stdout.end=0;
    stdout.array=array_out;
    stdout.size=BUFFER_SIZE;
    add_in_out(1,&stdout);
        screen_write("\x1B[2J");
        //screen_write("\x1B[34;47m");
}
```

screen.h

```
/**
 * screen.h | Interfaz para manejo de pantalla.
 */
#include "../system/common.h"

#ifndef SCREEN_H
#define SCREEN_H
/**
 * Escribe un caracter en pantalla.
 * @param char c: el caracter a escribir.
 *  Scape Characters implementados:
```

```
 *     Esc[2J                   Erase Display: Clears the screen and ↩
     moves the cursor to the home position (line 0, column 0).
 *
 *     Esc[#;#;...m      Set Graphics Mode: Calls the graphics ↩
     functions specified by the following values. These specified ↩
     functions remain active until the next occurrence of this escape ↩
     sequence. Graphics mode changes the colors and attributes of text↩
      (such as bold and underline) displayed on the screen.
 *
 * Text attributes
 * 0     All attributes off
 * 1     Bold on
 * 4     Underscore (on monochrome display adapter only)
 * 5     Blink on
 *
 * Foreground colors     Background colors
 * 30    Black             40   Black
 * 31    Red               41   Red
 * 32    Green             42   Green
 * 33    Yellow            43   Yellow
 * 34    Blue              44       Blue
 * 35    Magenta           45       Magenta
 * 36    Cyan              46       Cyan
 * 37    White             47       White
 *
 * Ej: Esc[34;47m (azul en fondo blanco)
 **/
void screen_put(char c);


#endif
```

timer.c

```
#include "../system/isr.h"
#include "../system/int80.h"

#define SUB_FUNC_VEC_SIZE 10

int80_t sub_handler_vec[SUB_FUNC_VEC_SIZE];

int ticks;
int count_ticks;
int sub_func_count;

void register_tick_subhandler(int80_t func) {
    if(sub_func_count<SUB_FUNC_VEC_SIZE-1){
            sub_handler_vec[sub_func_count] = func;
        sub_func_count++;
    }
}


void IRQ0_handler(registers_t regs){
    int i;
    if(count_ticks){
        ticks++;
    }
    for(i=0;i<sub_func_count;i++){
        sub_handler_vec[i](regs);
    }
}

void cpu_speed(registers_t regs){
    unsigned long k,t;
    count_ticks=1;
    ticks=0;
    _Sti();
    k=getRDTSC();
    while(ticks<30);
    k=getRDTSC()-k;
    _Cli();
```

```c
    count_ticks=0;
    *((unsigned long*)regs.ebx)=(k/ticks)*18+k/(ticks*5);
}

void init_timer_tick(){
    sub_func_count=0;
    count_ticks=0;
    register_interrupt_handler(IRQ0,IRQ0_handler);
    register_functionality(5,cpu_speed);
}
```

timer.h

```c
#include "../system/int80.h"

#ifndef TIMER_H
#define TIMER_H

void register_tick_subhandler(int80_t func);

void init_timer_tick();


void start_ticks();
void stop_ticks();
int get_ticks();
#endif  /* TIMER_H */
```

4.2.3. system

common.h

```c
#ifndef COMMON_H
#define COMMON_H

// Exact-width integer types
typedef   signed char   int8_t;
typedef unsigned char   uint8_t;
typedef   signed short  int16_t;
typedef unsigned short  uint16_t;
typedef   signed int    int32_t;
typedef unsigned int    uint32_t;

#define NULL ((void*)0)

// PIC
#define PORT_PIC1 0x20
#define PORT_PIC2 0xA0
#define SIGNAL_EOI 0x20

extern void outw(uint16_t port, uint16_t value);
extern void outb(uint16_t port, uint8_t value);
extern uint8_t inb(uint16_t port);
extern uint16_t inw(uint16_t port);
extern uint32_t getRDTSC();

#endif // COMMON_H
```

idt.c

```c
//
// descriptor_tables.c - Initialises the GDT and IDT, and defines the
//                       default ISR and IRQ handler.
//                    Based on code from Bran's kernel development tutorials.
//                    Rewritten for JamesM's kernel development tutorials.
//

#include "common.h"
#include "idt.h"
#include "isr.h"

// Lets us access our ASM functions from our C code.
extern void idt_flush(uint32_t);

// Internal function prototypes.
static void init_idt();
static void idt_set_gate(uint8_t, uint32_t, uint16_t, uint8_t);

idt_entry_t idt_entries[256];
idt_ptr_t   idt_ptr;

// Extern the ISR handler array so we can nullify them on startup.
extern isr_t interrupt_handlers[];

// Initialisation routine - zeroes all the interrupt service routines,
// initialises the GDT and IDT.
void init_descriptor_tables()
{
    /* Habilito interrupcion de timer tick*/
    _Cli();
    _mascaraPIC1(0xFE);
    _mascaraPIC2(0xFF);
    _Sti();

    // Initialise the interrupt descriptor table.
    init_idt();
}


static void init_idt()
{
    idt_ptr.limit = sizeof(idt_entry_t) * 256 -1;
    idt_ptr.base  = (uint32_t)&idt_entries;

    // Remap the irq table.
    outb(0x20, 0x11);
    outb(0xA0, 0x11);
    outb(0x21, 0x20);
    outb(0xA1, 0x28);
    outb(0x21, 0x04);
    outb(0xA1, 0x02);
    outb(0x21, 0x01);
    outb(0xA1, 0x01);
    outb(0x21, 0x0);
    outb(0xA1, 0x0);

    idt_set_gate( 0, (uint32_t)isr0 , 0x08, 0x8E);
    idt_set_gate( 1, (uint32_t)isr1 , 0x08, 0x8E);
    idt_set_gate( 2, (uint32_t)isr2 , 0x08, 0x8E);
    idt_set_gate( 3, (uint32_t)isr3 , 0x08, 0x8E);
    idt_set_gate( 4, (uint32_t)isr4 , 0x08, 0x8E);
    idt_set_gate( 5, (uint32_t)isr5 , 0x08, 0x8E);
    idt_set_gate( 6, (uint32_t)isr6 , 0x08, 0x8E);
    idt_set_gate( 7, (uint32_t)isr7 , 0x08, 0x8E);
    idt_set_gate( 8, (uint32_t)isr8 , 0x08, 0x8E);
    idt_set_gate( 9, (uint32_t)isr9 , 0x08, 0x8E);
    idt_set_gate(10, (uint32_t)isr10, 0x08, 0x8E);
    idt_set_gate(11, (uint32_t)isr11, 0x08, 0x8E);
    idt_set_gate(12, (uint32_t)isr12, 0x08, 0x8E);
    idt_set_gate(13, (uint32_t)isr13, 0x08, 0x8E);
    idt_set_gate(14, (uint32_t)isr14, 0x08, 0x8E);
    idt_set_gate(15, (uint32_t)isr15, 0x08, 0x8E);
    idt_set_gate(16, (uint32_t)isr16, 0x08, 0x8E);
    idt_set_gate(17, (uint32_t)isr17, 0x08, 0x8E);
    idt_set_gate(18, (uint32_t)isr18, 0x08, 0x8E);
    idt_set_gate(19, (uint32_t)isr19, 0x08, 0x8E);
    idt_set_gate(20, (uint32_t)isr20, 0x08, 0x8E);
    idt_set_gate(21, (uint32_t)isr21, 0x08, 0x8E);
    idt_set_gate(22, (uint32_t)isr22, 0x08, 0x8E);
    idt_set_gate(23, (uint32_t)isr23, 0x08, 0x8E);
    idt_set_gate(24, (uint32_t)isr24, 0x08, 0x8E);
    idt_set_gate(25, (uint32_t)isr25, 0x08, 0x8E);
    idt_set_gate(26, (uint32_t)isr26, 0x08, 0x8E);
    idt_set_gate(27, (uint32_t)isr27, 0x08, 0x8E);
    idt_set_gate(28, (uint32_t)isr28, 0x08, 0x8E);
    idt_set_gate(29, (uint32_t)isr29, 0x08, 0x8E);
    idt_set_gate(30, (uint32_t)isr30, 0x08, 0x8E);
    idt_set_gate(31, (uint32_t)isr31, 0x08, 0x8E);

    idt_set_gate(32, (uint32_t)irq0, 0x08, 0x8E);
    idt_set_gate(33, (uint32_t)irq1, 0x08, 0x8E);
    idt_set_gate(34, (uint32_t)irq2, 0x08, 0x8E);
    idt_set_gate(35, (uint32_t)irq3, 0x08, 0x8E);
    idt_set_gate(36, (uint32_t)irq4, 0x08, 0x8E);
    idt_set_gate(37, (uint32_t)irq5, 0x08, 0x8E);
    idt_set_gate(38, (uint32_t)irq6, 0x08, 0x8E);
    idt_set_gate(39, (uint32_t)irq7, 0x08, 0x8E);
    idt_set_gate(40, (uint32_t)irq8, 0x08, 0x8E);
    idt_set_gate(41, (uint32_t)irq9, 0x08, 0x8E);
    idt_set_gate(42, (uint32_t)irq10, 0x08, 0x8E);
    idt_set_gate(43, (uint32_t)irq11, 0x08, 0x8E);
    idt_set_gate(44, (uint32_t)irq12, 0x08, 0x8E);
    idt_set_gate(45, (uint32_t)irq13, 0x08, 0x8E);
    idt_set_gate(46, (uint32_t)irq14, 0x08, 0x8E);
    idt_set_gate(47, (uint32_t)irq15, 0x08, 0x8E);


    idt_set_gate(0X80, (uint32_t)isr80h, 0x08, 0x8E);


    idt_flush((uint32_t)&idt_ptr);
}

static void idt_set_gate(uint8_t num, uint32_t base, uint16_t sel, ↩
    uint8_t flags)
{
    idt_entries[num].base_lo = base & 0xFFFF;
    idt_entries[num].base_hi = (base >> 16) & 0xFFFF;

    idt_entries[num].sel     = sel;
    idt_entries[num].always0 = 0;
    // We must uncomment the OR below when we get to using user-mode.
    // It sets the interrupt gate's privilege level to 3.
    idt_entries[num].flags   = flags /* | 0x60 */;
}
```

idt.h

```
#include "common.h"

// Initialisation function is publicly accessible.
void init_descriptor_tables();

// A struct describing an interrupt gate.
struct idt_entry_struct
{
    uint16_t base_lo;             // The lower 16 bits of the address ↩
        to jump to when this interrupt fires.
    uint16_t sel;                 // Kernel segment selector.
    uint8_t  always0;             // This must always be zero.
    uint8_t  flags;               // More flags. See documentation.
    uint16_t base_hi;             // The upper 16 bits of the address ↩
        to jump to.
} __attribute__((packed));

typedef struct idt_entry_struct idt_entry_t;
```

```
// A struct describing a pointer to an array of interrupt handlers.
// This is in a format suitable for giving to 'lidt'.
struct idt_ptr_struct
{
    uint16_t limit;
    uint32_t base;                // The address of the first element in our idt_entry_t array.
} __attribute__((packed));

typedef struct idt_ptr_struct idt_ptr_t;

#define IDT_SIZE 256

// These extern directives let us access the addresses of our ASM ISR handlers.
extern void isr0 ();
extern void isr1 ();
extern void isr2 ();
extern void isr3 ();
extern void isr4 ();
extern void isr5 ();
extern void isr6 ();
extern void isr7 ();
extern void isr8 ();
extern void isr9 ();
extern void isr10();
extern void isr11();
extern void isr12();
extern void isr13();
extern void isr14();
extern void isr15();
extern void isr16();
extern void isr17();
extern void isr18();
extern void isr19();
extern void isr20();
extern void isr21();
extern void isr22();
extern void isr23();
extern void isr24();
extern void isr25();
extern void isr26();
extern void isr27();
extern void isr28();
extern void isr29();
extern void isr30();
extern void isr31();
extern void irq0 ();
extern void irq1 ();
extern void irq2 ();
extern void irq3 ();
extern void irq4 ();
extern void irq5 ();
extern void irq6 ();
extern void irq7 ();
extern void irq8 ();
extern void irq9 ();
extern void irq10();
extern void irq11();
extern void irq12();
extern void irq13();
extern void irq14();
extern void irq15();

extern void isr80h();
```

in_out.c

```
#include "int80.h"
#include "in_out.h"

buffer_t * in_out_vector[10];
```

```
void READ_INTERRUPT_handler(registers_t regs){
    int i;
    buffer_t * buff=in_out_vector[regs.ebx];
    for(i=0;i<regs.edx && buff->start!=buff->end;i++){
            *((char*)(regs.ecx+i))=buff->array[buff->start];
            buff->start=(buff->start+1)%buff->size;
    }
    if(i<regs.edx){
        *((char*)(regs.ecx+i))='\0';
    }
}

void WRITE_INTERRUPT_handler(registers_t regs){
    int i;
    int tmp;
    buffer_t * buff=in_out_vector[regs.ebx];
    tmp=(buff->end+1)%buff->size;
    for(i=0;i<regs.edx && tmp!=buff->start;i++,tmp=(buff->end+1)%buff ↩
        ->size){
        buff->array[buff->end]=*((char*)(regs.ecx+i));
        buff->end=tmp;
    }
}

void add_in_out(int n, buffer_t * buff){
    in_out_vector[n]=buff;
}


init_in_out(){
    register_functionality(3,READ_INTERRUPT_handler);
    register_functionality(4,WRITE_INTERRUPT_handler);
}
```

in_out.h

```
#ifndef IN_H
#define IN_H


struct buffer_struct
{
    int size;
    char * array;
    int start;
    int end;
};

typedef struct buffer_struct buffer_t;

#endif // IN_H
```

int80.c

```
#include "isr.h"
#include "int80.h"

#define SUB_FUNC_VEC_SIZE 10



int80_t sub_funcs_vec[SUB_FUNC_VEC_SIZE];


void register_functionality(uint8_t n, int80_t func) {
    if(n<SUB_FUNC_VEC_SIZE){
        sub_funcs_vec[n] = func;
    }
}

void int80_handler(registers_t regs){
    if(regs.eax<SUB_FUNC_VEC_SIZE){
        sub_funcs_vec[regs.eax](regs);
    }
}

void nofunc(registers_t regs){
}



void init_int80(){
    int i;
    for(i=0;i<SUB_FUNC_VEC_SIZE;i++){
        sub_funcs_vec[i]=nofunc;
    }
    register_interrupt_handler(0X80,int80_handler);
}
```

int80.h

```
#include "isr.h"

#ifndef INT80_H
#define INT80_H

typedef void (*int80_t)(registers_t);
void register_functionality(uint8_t n, int80_t func);
void init_int80();

#endif  /* INT80_H */
```

isr.c

```
#include "common.h"
#include "isr.h"
#include "idt.h"

isr_t interrupt_handlers[IDT_SIZE];

void register_interrupt_handler(uint8_t n, isr_t handler) {
    interrupt_handlers[n] = handler;
}

void isr_handler(registers_t regs) {
    if(regs.int_no==-128){//cableo orrendo, pero por alguna razon me ↩
        lo pone negativo
        regs.int_no*=-1;
        }
    if (interrupt_handlers[regs.int_no] != NULL) {
        isr_t handler = interrupt_handlers[regs.int_no];
        handler(regs);
    }
}

void irq_handler(registers_t regs) {
    if (regs.int_no >= IRQ8) {
        outb(PORT_PIC2, SIGNAL_EOI);
    }
    outb(PORT_PIC1, SIGNAL_EOI);
    isr_handler(regs);
}
```

isr.h

```
#include "common.h"

#ifndef ISR_H
#define ISR_H

// A few defines to make life a little easier
#define IRQ0 32
#define IRQ1 33
#define IRQ2 34
#define IRQ3 35
#define IRQ4 36
#define IRQ5 37
#define IRQ6 38
#define IRQ7 39
#define IRQ8 40
#define IRQ9 41
#define IRQ10 42
#define IRQ11 43
#define IRQ12 44
#define IRQ13 45
#define IRQ14 46
#define IRQ15 47

typedef struct registers
{
    uint32_t ds;                  // Data segment selector
    uint32_t edi, esi, ebp, esp, ebx, edx, ecx, eax; // Pushed by pusha.
    uint32_t int_no, err_code;    // Interrupt number and error code (if applicable)
    uint32_t eip, cs, eflags, useresp, ss; // Pushed by the processor automatically.
} registers_t;

// Enables registration of callbacks for interrupts or IRQs.
// For IRQs, to ease confusion, use the #defines above as the
// first parameter.
typedef void (*isr_t)(registers_t);
void register_interrupt_handler(uint8_t n, isr_t handler);

#endif //ISR_H
```

keyboardlisteners.c

```
#ifndef KEYBOARDLISTENER_H
#define KEYBOARDLISTENER_H

#define MAX_SCAN_CODE 300

#define CTRL_KEY_PRESED_SCAN_CODE 29
#define CTRL_KEY_RELESED_SCAN_CODE 157

#define ALT_KEY_PRESED_SCAN_CODE 56
#define ALT_KEY_RELESED_SCAN_CODE 184

typedef int (*key_listener)();

int activate(int scan_code);
void add_key_listener(int mode, int scan_code, key_listener listener);
void init_key_listeners();

#endif //KEYBOARDLISTENER_H
```

keyboardlisteners.h

```
#ifndef KEYBOARDLISTENER_H
#define KEYBOARDLISTENER_H
```

```
#define MAX_SCAN_CODE 300

#define CTRL_KEY_PRESED_SCAN_CODE 29
#define CTRL_KEY_RELESED_SCAN_CODE 157

#define ALT_KEY_PRESED_SCAN_CODE 56
#define ALT_KEY_RELESED_SCAN_CODE 184

typedef int (*key_listener)();

int activate(int scan_code);
void add_key_listener(int mode, int scan_code, key_listener listener);
void init_key_listeners();

#endif //KEYBOARDLISTENER_H
```

4.2.4. asm

idt.asm

```
[GLOBAL idt_flush]    ; Allows the C code to call idt_flush().

idt_flush:
    mov eax, [esp+4]  ; Get the pointer to the IDT, passed as a ↩
        parameter.
    lidt [eax]        ; Load the IDT pointer.
    ret

%macro ISR_NOERRCODE 1
  global isr%1
  isr%1:
    cli                         ; Disable interrupts firstly.
    push byte 0                 ; Push a dummy error code.
    push byte %1                ; Push the interrupt number.
    jmp isr_common_stub         ; Go to our common handler code.
%endmacro

; This macro creates a stub for an ISR which passes it's own
; error code.
%macro ISR_ERRCODE 1
  global isr%1
  isr%1:
    cli                         ; Disable interrupts.
    push byte %1                ; Push the interrupt number
    jmp isr_common_stub
%endmacro

; This macro creates a stub for an IRQ - the first parameter is
; the IRQ number, the second is the ISR number it is remapped to.
%macro IRQ 2
  global irq%1
  irq%1:
    cli
    push byte 0
    push byte %2
    jmp irq_common_stub
%endmacro

ISR_NOERRCODE 0
ISR_NOERRCODE 1
ISR_NOERRCODE 2
ISR_NOERRCODE 3
ISR_NOERRCODE 4
ISR_NOERRCODE 5
ISR_NOERRCODE 6
ISR_NOERRCODE 7
ISR_ERRCODE   8
ISR_NOERRCODE 9
ISR_ERRCODE   10
ISR_ERRCODE   11
ISR_ERRCODE   12
ISR_ERRCODE   13
ISR_ERRCODE   14
ISR_NOERRCODE 15
ISR_NOERRCODE 16
ISR_NOERRCODE 17
ISR_NOERRCODE 18
ISR_NOERRCODE 19
ISR_NOERRCODE 20
ISR_NOERRCODE 21
ISR_NOERRCODE 22
ISR_NOERRCODE 23
ISR_NOERRCODE 24
ISR_NOERRCODE 25
ISR_NOERRCODE 26
ISR_NOERRCODE 27
ISR_NOERRCODE 28
ISR_NOERRCODE 29
ISR_NOERRCODE 30
ISR_NOERRCODE 31

IRQ   0,    32
IRQ   1,    33
IRQ   2,    34
IRQ   3,    35
IRQ   4,    36
IRQ   5,    37
IRQ   6,    38
IRQ   7,    39
IRQ   8,    40
IRQ   9,    41
IRQ  10,    42
IRQ  11,    43
IRQ  12,    44
IRQ  13,    45
IRQ  14,    46
IRQ  15,    47

  global isr80h
  isr80h:
    cli                          ; Disable interrupts firstly.
    push byte 0                  ; Push a dummy error code.
    push byte 128                ; Push the interrupt number.
    jmp isr_common_stub          ; Go to our common handler code.


; In isr.c
extern isr_handler

; This is our common ISR stub. It saves the processor state, sets
; up for kernel mode segments, calls the C-level fault handler,
; and finally restores the stack frame.
isr_common_stub:
    pusha                     ; Pushes edi,esi,ebp,esp,ebx,edx,ecx,eax

    mov ax, ds                ; Lower 16-bits of eax = ds.
    push eax                  ; save the data segment descriptor

    mov ax, 0x10  ; load the kernel data segment descriptor
    mov ds, ax
    mov es, ax
    mov fs, ax
    mov gs, ax

    call isr_handler

    pop ebx        ; reload the original data segment descriptor
    mov ds, bx
    mov es, bx
    mov fs, bx
    mov gs, bx

    popa                      ; Pops edi,esi,ebp...
    add esp, 8      ; Cleans up the pushed error code and pushed ISR ↩
        number
    sti
    iret            ; pops 5 things at once: CS, EIP, EFLAGS, SS, and ↩
        ESP

; In isr.c
extern irq_handler

; This is our common IRQ stub. It saves the processor state, sets
; up for kernel mode segments, calls the C-level fault handler,
; and finally restores the stack frame.
irq_common_stub:
    pusha                     ; Pushes edi,esi,ebp,esp,ebx,edx,ecx,eax

    mov ax, ds                ; Lower 16-bits of eax = ds.
    push eax                  ; save the data segment descriptor

    mov ax, 0x10  ; load the kernel data segment descriptor
    mov ds, ax
    mov es, ax
    mov fs, ax
    mov gs, ax

    call irq_handler

    pop ebx         ; reload the original data segment descriptor
    mov ds, bx
    mov es, bx
    mov fs, bx
    mov gs, bx

    popa                      ; Pops edi,esi,ebp...
    add esp, 8      ; Cleans up the pushed error code and pushed ISR ↩
        number
    sti
    iret            ; pops 5 things at once: CS, EIP, EFLAGS, SS, and ↩
        ESP
```

common.asm

```
global outb
global outw
global inb
global inw
global getRDTSC

getRDTSC:
    rdtsc
    ret

outb:
    mov dx, [esp+4]
    mov al, [esp+8]
    out dx, al
    ret

outw:
    mov dx, [esp+4]
    mov ax, [esp+8]
    out dx, ax
    ret

inb:
    mov dx, [esp+4]
    in al, dx
    ret

inw:
    mov dx, [esp+4]
    in ax, dx
    ret
```

4.2.5. std

getchar.c

```
#include "stdio.h"

#define STREAM_SIZE 500

typedef int (*flusher)(char * streampointer);

char stream[STREAM_SIZE];
char * streamout=stream;


int intro_flush(char * streampointer){
    if(*streampointer=='\n' || 1>=STREAM_SIZE-(streampointer-stream)↩
        -1){
        return 1;
    }

    return 0;
}

char getchar(){
    char c=*streamout;
    if(c=='\0'){
        streamout=stream;
        char * streamin=stream;
        int i,j;
        for(i=0;i<STREAM_SIZE;i++){
            stream[i]='\0';
        }
        while(!intro_flush(streamin)){
            if(*streamin!='\0')
                streamin++;
            __read(0,streamin,1);
            if(*streamin!='\b' && *streamin!='\t'){
                printf(streamin);
            }
            else if(*streamin=='\b'){
                if(streamin > stream){
                    printf("\b");
                    *streamin='\0';
                    streamin--;
                }
                *streamin='\0';
                        } else if(*streamin=='\t'){
                            *streamin='\0';
                        }
        }
        c=*streamout;
    }
    streamout++;
    return c;

}
```

printf.c

```
#include "stdio.h"

static void prints(char * string);

static char * numberBaseNtoString(unsigned int number, int base, char ↩
    * out);
```

```
void putchar(char c){
    __write(1,&c,1);
}

void printf( char * formatString, ...) {
    int integer;
    unsigned int unsigenedInteger;
    char * string;
    char out[40];

    va_list args;

    va_start(args, formatString);

    while( *formatString !='\0' )
    {
        if(*formatString == '%'){

            formatString++;

            switch(*formatString){
            case 'c' :
                  integer = va_arg(args,char);
                 putchar(integer);
                  break;
            case 's':
                  string = va_arg(args,char *);
                  prints(string);
                  break;
            case 'd' :
                  integer = va_arg(args,int);
                  if(integer < 0){
                      integer = -integer;
                      putchar('-');
                  }
                  prints(numberBaseNtoString(integer,10,out));
                  break;
            case 'u':
                  unsigenedInteger = va_arg(args, unsigned int);
                  prints(numberBaseNtoString(unsigenedInteger,10,out))↩
                      ;
                  break;
            case 'o':
                  integer = va_arg(args,unsigned int);
                  prints(numberBaseNtoString(integer,8,out));
                  break;
            case 'x':
                  unsigenedInteger = va_arg( args, unsigned int);
                  prints(numberBaseNtoString(unsigenedInteger,16,out))↩
                      ;
                  break;
            case '%':
                  putchar('%');
                  break;
            }
        } else{
            putchar(*formatString);
        }
        formatString++;
    }
    va_end(args);
}

static void prints(char * string){
    while(*string != '\0'){
        putchar(*string);
        string++;
    }
}


static char * numberBaseNtoString(unsigned int number, int base, char ↩
    * out){
```

```
    int digits[40];
    int position = 0;
    char * numbers = "0123456789ABCDEF";
    int index = 0;

    if( number != 0 ){
        while( number > 0 ){
            if(number < base) {
                  digits[position] = number;
                  number = 0;
            } else {
                  digits[position] = number % base;
                  number /= base;
            }
            position++;
        }

        for( index = 0 ; position > 0 ; position--, index++ ){
              out[index] = numbers[digits[position-1] % base];
        }
        out[index] = '\0';
    } else {
        out[0] = '0';
        out[1] = '\0';
    }

    return out;
}
```

scanf.c

```
#ifndef SCANF_C
#define SCANF_C

#include "printf.c"
#include "../../src/std/string.h"
#include "../../include/varargs.h"
#include "../../include/stdarg.h"

int( readFromStr)(char *formatString, char *format, ...) {
    va_list ap;
    va_start ( ap, format );
    float *f;
    int conv = 0, *integer, index, resp = 0;
    char *a, *fp, *sp = formatString, buf[256] = { '\0' };

    for (fp = formatString; *fp != '\0'; fp++) {
        for (index = 0; *sp != '\0' && *sp != ' '; index++) {
            buf[index] = *sp++;
        }
        buf[index] = '\0';
        while (*sp == ' ') {
            sp++;
        }
        while (*fp != '%') {
            fp++;
        }
        if (*fp == '%') {
            switch (*++fp) {
            case 'd':
                integer = va_arg ( ap, int * );
                for (j = 0; *fp != '\0' && *fp != ' '; fp++, j++) {
                    resp += ((*fp) - '0') * (10 ^ j);
                }
                *integer = resp;
                break;
            case 's':
                a = va_arg ( ap, char * );
                strncpy(buf, a);
                break;
            }
```

```
                conv++;
            }
        }
        va_end ( ap );
        return conv;
}
```

stdio.h

```
#include "../../include/varargs.h"
#include "../../include/stdarg.h"

#ifndef STDIO_H
#define STDIO_H

char getchar();
void putchar(char c);
void printf( char * formatString, ...);

#endif //STDIO_H
```

string.c

```

int strcmp(char* str1, char * str2){
    int i;
    for(i=0;str1[i]!='\0' && str1[i]!='\0' ;i++){
        if(str1[i]!=str2[i]){
            return str1[i]-str2[i];
        }
    }
    if(str1[i]=='\0' && str2[i]=='\0'){
        return str1[i]-str2[i];
    }
    return 1;
}


void strcpy(char * str_des,char * str_ori){
    int i;
    for(i=0;str_ori[i]!='\0';i++){
        str_des[i]=str_ori[i];
    }
    str_des[i]='\0';
}

int strlen(char* str){
    int i;
    for(i=0;str[i]!='\0';i++);
    return i;
}
```

string.h

```
#ifndef STRING_H
#define STRING_H

int strcmp(char* str1, char * str2);
void strcpy(char * str_des,char * str_ori);
int strlen(char* str);

#endif /* STRING_H */
```

systemcall.asm

```
global __read
global __write
global __cpuspeed

SECTION .text

__read:
        mov ecx, [esp+8]
    mov eax,3
    mov ebx, [esp+4]
    mov edx, [esp+12]
    int 80h
    ret

__write:
        mov ecx, [esp+8]
    mov eax,4
    mov ebx, [esp+4]
    mov edx, [esp+12]
    int 80h
    ret

__cpuspeed:
     mov ebx, [esp+4]
     mov eax,5
     int 80h
     ret
```

systemcall.h

```
#ifndef SYSTEMCALL_H
#define SYSTEMCALL_H

void __read(int fd, void* buffer, int count);
void __write(int fd, const void* buffer, int count);
void __cpuspeed(void * ips);

#endif  /* SYSTEMCALL_H */
```

4.2.6. user

commands.c

```
#include "commands.h"

#include "../std/string.h"

#define NULL 0
#define COMMAND_MAX_CANT 20

command_t command_list[COMMAND_MAX_CANT];
int commands_added=0;

char** get_command_list() {
    char* commands[COMMAND_MAX_CANT];
    int i;
    for(i=0;i<commands_added;i++) {
        commands[i] = command_list[i].name;
    }
    commands[i] = NULL;
    return commands;
}

void add_command(char * name,main function){
    if(commands_added<COMMAND_MAX_CANT){
        command_list[commands_added].name=name;
        command_list[commands_added].start=function;
        commands_added++;
    }
}

main get_command(char * name){
    int i;
    for(i=0;i<commands_added;i++){
        if(!strcmp(command_list[i].name,name)){
            return command_list[i].start;
        }
    }
    return NULL;
}
```

commands.h

```
#ifndef COMMANDS_H
#define COMMANDS_H

typedef int (*main)(int argc,char * argv[]);


struct command_struct {
    char * name;
    main start;
};

typedef struct command_struct command_t;

void add_command(char * name,main function);
main get_command(char * name);

char * autocomplete(char * name);

#endif //COMMANDS_H
```

shell.c

```
#include "shell.h"

#include "../std/stdio.h"
#include "../std/string.h"

#include "commands.h"

#define NULL 0
#define COMAND_LINE_MAX 1000

#define HISTORY_MAX 20
char* history[HISTORY_MAX][COMAND_LINE_MAX];
int history_current = 0;
int history_count = 0;

char * name="user";
char * pcname="thispc";


char * strnormalise(char * str){
        int j, i;
        // cambia enters por espacios
        for(j=0;str[j]!='\0';j++){
                if(str[j]=='\n' || str[j] == '\t'){
                        str[j]=' ';
                }
        }
        // elimina espacios del principio
        while(str[0]==' '){
                str=str+1;
        }
        //elimina espacios del final
        for(i=strlen(str)-1;i>0 && str[i]==' ';i--){
                str[i]='\0';
        }
        //elimina espacios repetidos en el medio
        for(j=0;str[j]!='\0';j++){
                if(str[j]==' ' && str[j+1]==' '){
                        strcpy(str + j, str + j + 1);
                        j--;
                }
        }
        return str;
}

void printuser(){
    printf("\x1B[32m%s@%s:\x1B[0m",name,pcname);
}

int execute(char* comand,int argcant,char * argvec[]){
    if(comand[0]=='\0'){
        return 0;
    }
    main start=get_command(comand);
    if(start==NULL){
        printf("invalid comand: %s\n",comand);
        return -1;
    }
    return start(argcant,argvec);
}

int parseline(){
    char c;
    int i=0;
    char comand_line[COMAND_LINE_MAX];
    while((c=getchar())!='\n' && i<COMAND_LINE_MAX-3){
        comand_line[i]=c;
        i++;
    }
    if(i>=COMAND_LINE_MAX-3){
        while(getchar()!='\n');
        printf("\n");
    }
    comand_line[i]='\0';
    char* command=strnormalise(comand_line);
    int argcant=0;
    char * argvec[50];
        int in_quotes = 0;
    for(i=0;command[i]!='\0';i++){
        if(command[i]==' ' && !in_quotes){
            command[i]='\0';
            argvec[argcant]=&command[i+1];
            argcant++;
                } else if (command[i]=='"') {
                    in_quotes = !in_quotes;
                }
    }
    return execute(command,argcant,argvec)==-15;
}

int exit_shell(int argc,char* argv[]){
      return -15;
}

int echo_shell(int argc,char* argv[]){
    int i;
    for(i=0;i<argc;i++){
        printf("%s\n",argv[i]);
    }
    return 0;
}

int getCPUspeed_shell(){
    unsigned long ips;
    __cpuspeed(&ips);
    //printf("Su procesador esta ejecutando %d instrucciones por ↩
        segudo.\n",ips);
    printf("La velocidad en MHz es:%d.%d MHz\n",(ips)/(1024*1024)↩
        ,((10*ips)/(1024*1024))%10);
    return 0;
}

int clear_shell(){
    printf("\x1B[2J");
    return 0;
}

int help_shell(){
    printf("These are the commands available: \n");
    char** commands = get_command_list();
    int i = 0;
    while(commands[i]) {
        printf("\x1B[4m%s\x1B[0m\t\n", commands[i++]);
    }
    return 0;
}

void shell_start(){
    int exit=0;
    add_command("echo", echo_shell);
    add_command("exit", exit_shell);
    add_command("getCPUspeed", getCPUspeed_shell);
    add_command("clear", clear_shell);
        add_command("help", help_shell);
    while(!exit)
    {
        printuser();
        exit=parseline();
    }
}
```

shell.h

```
#ifndef SHELL_H
#define SHELL_H

void shell_start();

#endif  /* SHELL_H */
```

5. Referencias

Esta sección detalla las distintas fuentes de información utilizadas para el desarrollo del TP Especial.

5.1. Interrupciones

El manejo de interrupciones es similar al usado en el siguiente tutorial:

http://www.jamesmolloy.co.uk/tutorial_html/4.-
The%20GDT%20and%20IDT.html

Nos pareció interesante la opción de crear un wrapper para las idt y luego desde C simplemente asignar handlers a las convenientes, un wrapper se encarga de que a C le lleguen los parametros.También creimos importante tener las entrys para las primeras 31 exceptions del procesador, para evitar resets si por ejemplo dividimos por cero.

5.2. Pantalla

Nos basamos en la implementación de Linux, la pantalla puede recibir scape chars para limpiar la pantalla, o imprimir en colores. Esto era conveniente debido a que al utilizar la int80, no necesitamos parametros extra, para estas funcionalidades extra.

5.3. Reboot

Luego de probar soluciones sucias, como hacer que el procesador triple-faultee y se reinicie la pc. Encontrámos la solucion de enviar la señal de reset desde el controlador de teclado en:

http://wiki.osdev.org/Reboot

5.4. Paginas web utilizadas

Estos son los puntos de encuentro mas fuertes, esta información fue informada por mail previo a la entrega. En generál leimos bastante de:

http://wiki.osdev.org/Main_Page (y sus foros)
http://www.jamesmolloy.co.uk/tutorial_html/index.html
http://www.osdever.net/tutorials/view/brans-kernel-development-
tutorial