

Codigos fuente

Arnix

Modo protegido con GRUB

31 de mayo de 2011

Autores:

<i>Axel Wassington</i>	Legajo: 50124
<i>Horacio Miguel Gomez</i>	Legajo: 50825
<i>Tomás Mehdi</i>	Legajo: 51014

Índice

1. Codigos fuente	3
1.1. defs.h	3
1.2. kasm.h	4
1.3. kc.h	4
1.4. stdarg.h	4
1.5. varargs.h	6
1.6. kernel.c	7
1.7. lib.asm	8
1.8. lib.c	9
1.9. loader.asm	9
1.10. keyboard.c	10
1.11. keyboard.h	12
1.12. screen.c	12
1.13. screen.h	15
1.14. timer.c	16
1.15. timer.h	17
1.16. common.h	17
1.17. idt.c	18
1.18. idt.h	19
1.19. in_out.c	20
1.20. in_out.h	21
1.21. int80.c	21
1.22. int80.h	22
1.23. isr.c	22
1.24. isr.h	23
1.25. keyboardlisteners.c	23
1.26. keyboardlisteners.h	24
1.27. idt.asm	24
1.28. common.asm	26
1.29. getchar.c	27
1.30. printf.c	28
1.31. scanf.c	29
1.32. stdio.h	30
1.33. string.c	30
1.34. string.h	31
1.35. syscall.asm	31
1.36. syscall.h	32
1.37. commands.c	32
1.38. commands.h	33
1.39. shell.c	33

1.40. shell.h	36
-------------------------	----

1. Codigos fuente

include

1.1. defs.h

```

1  /******
2  Defs.h
3
4  *****/
5
6  #ifndef _defs_
7  #define _defs_
8
9  #define byte unsigned char
10 #define word short int
11 #define dword int
12
13 /* Flags para derechos de acceso de los segmentos */
14 #define ACS_PRESENT 0x80 /* segmento presente en ←
    memoria */
15 #define ACS_CSEG 0x18 /* segmento de codigo */
16 #define ACS_DSEG 0x10 /* segmento de datos */
17 #define ACS_READ 0x02 /* segmento de lectura */
18 #define ACS_WRITE 0x02 /* segmento de escritura */
19 #define ACS_IDT ACS_DSEG
20 #define ACS_INT_386 0x0E /* Interrupt GATE 32 bits */
21 #define ACS_INT ( ACS_PRESENT | ACS_INT_386 )
22
23
24 #define ACS_CODE (ACS_PRESENT | ACS_CSEG | ACS_READ)
25 #define ACS_DATA (ACS_PRESENT | ACS_DSEG | ACS_WRITE)
26 #define ACS_STACK (ACS_PRESENT | ACS_DSEG | ACS_WRITE)
27
28 #pragma pack (1) /* Alinear las siguiente estructuras a 1 byte ←
    */
29
30 /* Descriptor de segmento */
31 typedef struct {
32     word limit,
33     base_l;
34     byte base_m,
35     access,
36     attribs,
37     base_h;
38 } DESCR_SEG;
39
40
41 /* Descriptor de interrupcion */
42 typedef struct {
43     word offset_l,
44     selector;
45     byte cero,
46     access;
47     word offset_h;
48 } DESCR_INT;
49
50 /* IDTR */
51 typedef struct {
52     word limit;
53     dword base;
54 } IDTR;
55
56
57
58 #endif

```

1.2. kasm.h

```
1  /******  
2  kasm.h  
3  *****/  
4  *****/  
5  *****/  
6  #include "defs.h"  
7  
8  
9  unsigned int    _read_msw();  
10  
11 void            _lidt (IDTR *idtr);  
12  
13 void            _mascaraPIC1 (byte mascara); /* Escribe mascara de PIC1 ↵  
14 /*  
15 void            _mascaraPIC2 (byte mascara); /* Escribe mascara de PIC2 ↵  
16 /*  
17 void            _Cli(void); /* Deshabilita interrupciones */  
18 void            _Sti(void); /* Habilita interrupciones */  
19 void            _int_08_hand(); /* Timer tick */  
20  
21 void            _debug (void);
```

1.3. kc.h

```
1  /******  
2  kc.h  
3  *****/  
4  *****/  
5  *****/  
6  #ifndef _kc_  
7  #define _kc_  
8  
9  #define WHITE_TXT 0x07 // Atributo de video. Letras blancas, fondo ↵  
10 negro  
11  
12 /* Muestra la imagen de inicio */  
13 void showSplashScreen();  
14  
15 /* Tiempo de espera */  
16 void wait(int time);  
17  
18 /* Limpia la pantalla */  
19 void k_clear_screen();  
20  
21 /* Inicializa la entrada del IDT */  
22 void setup_IDT_entry (DESCR_INT *item, byte selector, dword offset, ↵  
23 byte access, byte cero);  
24 #endif
```

1.4. stdarg.h

```
1  /*  
2  * stdarg.h  
3  */
```

```

4  * Provides facilities for stepping through a list of function ↵
   * arguments of
5  * an unknown number and type.
6  *
7  * NOTE: Gcc should provide stdarg.h, and I believe their version will↵
   * work
8  * with crt.dll. If necessary I think you can replace this with ↵
   * the GCC
9  * stdarg.h (or is it vararg.h).
10 *
11 * Note that the type used in va_arg is supposed to match the actual ↵
   * type
12 * *after default promotions*. Thus, va_arg(..., short) is not valid.
13 *
14 * This file is part of the Mingw32 package.
15 *
16 * Contributors:
17 * Created by Colin Peters <colin@bird.fu.is.saga-u.ac.jp>
18 *
19 * THIS SOFTWARE IS NOT COPYRIGHTED
20 *
21 * This source code is offered for use in the public domain. You may
22 * use, modify or distribute it freely.
23 *
24 * This code is distributed in the hope that it will be useful but
25 * WITHOUT ANY WARRANTY. ALL WARRANTIES, EXPRESS OR IMPLIED ARE ↵
   * HEREBY
26 * DISCLAIMED. This includes but is not limited to warranties of
27 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
28 *
29 * $Revision: 1.1.1.1 $
30 * $Author: brandon6684 $
31 * $Date: 2001/12/18 22:53:51 $
32 *
33 */
34 /* Appropriated for Reactos Crt.dll by Ariadne */
35
36 #ifndef STDARG_H
37 #define STDARG_H
38
39 /*
40 * Don't do any of this stuff for the resource compiler.
41 */
42 #ifndef RC_INVOKED
43
44 /*
45 * I was told that Win NT likes this.
46 */
47 #ifndef _VA_LIST_DEFINED
48 #define _VA_LIST_DEFINED
49 #endif
50
51 #ifndef _VA_LIST
52 #define _VA_LIST
53 typedef char* va_list;
54 #endif
55
56
57 /*
58 * Amount of space required in an argument list (ie. the stack) for an
59 * argument of type t.
60 */
61 #define __va_argsiz(t) \
62     (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))
63
64
65 /*
66 * Start variable argument list processing by setting AP to point to ↵
   * the
67 * argument after pN.
68 */
69 #ifdef __GNUC__
70 /*

```

```

71  * In GNU the stack is not necessarily arranged very neatly in order ↵
72  to
73  * pack shorts and such into a smaller argument list. Fortunately a
74  * neatly arranged version is available through the use of ↵
75  __builtin_next_arg.
76  */
77 #define va_start(ap, pN) \
78 ((ap) = ((va_list) __builtin_next_arg(pN)))
79 #else
80 /*
81  * For a simple minded compiler this should work (it works in GNU too ↵
82  for
83  * vararg lists that don't follow shorts and such).
84  */
85 #define va_start(ap, pN) \
86 ((ap) = ((va_list) (&pN) + __va_argsiz(pN)))
87 #endif
88
89 /*
90  * End processing of variable argument list. In this case we do ↵
91  nothing.
92  */
93 #define va_end(ap) ((void)0)
94
95 /*
96  * Increment ap to the next argument in the list while returning a
97  * pointer to what ap pointed to first, which is of type t.
98  *
99  * We cast to void* and then to t* because this avoids a warning about
100  * increasing the alignment requirement.
101  */
102 #define va_arg(ap, t) \
103 ((ap) = (ap) + __va_argsiz(t)), \
104 *((t*) (void*) ((ap) - __va_argsiz(t)))
105 #endif /* Not RC_INVOKED */
106
107 #endif /* not _STDARG_H_ */

```

1.5. varargs.h

```

1  /* $NetBSD: varargs.h,v 1.11 2005/12/11 12:16:16 christos Exp $ */
2
3  /*-
4   * Copyright (c) 1990, 1993
5   * The Regents of the University of California. All rights reserved.
6   * (c) UNIX System Laboratories, Inc.
7   * All or some portions of this file are derived from material ↵
8   licensed
9   * to the University of California by American Telephone and Telegraph
10  * Co. or Unix System Laboratories, Inc. and are reproduced herein ↵
11  with
12  * the permission of UNIX System Laboratories, Inc.
13  *
14  * Redistribution and use in source and binary forms, with or without
15  * modification, are permitted provided that the following conditions
16  * are met:
17  * 1. Redistributions of source code must retain the above copyright
18  * notice, this list of conditions and the following disclaimer.
19  * 2. Redistributions in binary form must reproduce the above ↵
20  copyright
21  * notice, this list of conditions and the following disclaimer in ↵
22  the
23  * documentation and/or other materials provided with the ↵
24  distribution.

```

```

20 * 3. Neither the name of the University nor the names of its ↵
    contributors
21 * may be used to endorse or promote products derived from this ↵
    software
22 * without specific prior written permission.
23 *
24 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' ↵
    AND
25 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, ↵
    THE
26 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR ↵
    PURPOSE
27 * ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE ↵
    LIABLE
28 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR ↵
    CONSEQUENTIAL
29 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE ↵
    GOODS
30 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS ↵
    INTERRUPTION)
31 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, ↵
    STRICT
32 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ↵
    ANY WAY
33 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY ↵
    OF
34 * SUCH DAMAGE.
35 *
36 * @(#)varargs.h      8.2 (Berkeley) 3/22/94
37 */
38
39 #ifndef VARARGS_H
40 #define VARARGS_H
41
42 #if !__GNUC_PREREQ__
43 #define __va_ellipsis
44 #else
45 #define __va_ellipsis ...
46 #endif
47
48 #if __GNUC_PREREQ__
49 #define __va_alist_t __builtin_va_alist_t
50 #else
51 #define __va_alist_t long
52 #endif
53
54 #define va_alist __builtin_va_alist
55 #define va_dcl __va_alist_t __builtin_va_alist; __va_ellipsis
56
57
58 #endif

```

src

kernel

1.6. kernel.c

```

1 #include "../include/kasm.h"
2 #include "kernel/driver/screen.h"
3 #include "kernel/system/idt.h"
4 #include "kernel/driver/keyboard.h"
5 #include "kernel/system/keyboardlisteners.h"
6
7 /*****
8 kmain()
9 Punto de entrada de codigo C.
10 *****/
11

```



```
12 kmain()
13 {
14     // Init system.
15     init_descriptor_tables();
16     init_int80();
17     init_in_out();
18     init_keyboard();
19     init_timer_tick();
20     init_screen();
21
22     // Start Shell
23     shell_start();
24 }
```

1.7. lib.asm

```
1 GLOBAL _read_msw, _lidt
2 GLOBAL _int_08_hand
3 GLOBAL _mascaraPIC1, _mascaraPIC2, _Cli, _Sti
4 GLOBAL _debug
5
6 EXTERN int_08
7
8
9 SECTION .text
10
11
12
13 _Cli:
14     cli                ; limpia flag de interrupciones
15     ret
16
17 _Sti:
18
19     sti                ; habilita interrupciones por flag
20     ret
21
22 _mascaraPIC1:          ; Escribe mascara del PIC 1
23     push    ebp
24     mov     ebp, esp
25     mov     ax, [ss:ebp+8] ; ax = mascara de 16 bits
26     out     21h, al
27     pop     ebp
28     retn
29
30 _mascaraPIC2:          ; Escribe mascara del PIC 2
31     push    ebp
32     mov     ebp, esp
33     mov     ax, [ss:ebp+8] ; ax = mascara de 16 bits
34     out     0A1h, al
35     pop     ebp
36     retn
37
38 _read_msw:
39     smsw    ax          ; Obtiene la Machine Status Word
40     retn
41
42
43 _lidt:                ; Carga el IDTR
44     push    ebp
45     mov     ebp, esp
46     push    ebx
47     mov     ebx, [ss:ebp + 6] ; ds:bx = puntero a IDTR
48     rol     ebx, 16
49     lidt    [ds:ebx]      ; carga IDTR
50     pop     ebx
51     pop     ebp
52     retn
53
```

```
54
55
56
57 ; Debug para el BOCHS, detiene la ejecucion para continuar ; colocar ←
   en el BOCHSDBG: set $eax=0
58 _debug:
59     push    bp
60     mov     bp, sp
61     push    ax
62 vuelve: mov     ax, 1
63     cmp     ax, 0
64     jne     vuelve
65     pop     ax
66     pop     bp
67     retn
```

1.8. lib.c

```
1  #include "../include/kc.h"
2
3
4  /******
5  *k_clear_screen
6  *
7  * Borra la pantalla en modo texto color.
8  *****/
9
10 void k_clear_screen()
11 {
12     char *vidmem = (char *) 0xb8000;
13     unsigned int i=0;
14     while(i < (80*25*2))
15     {
16         vidmem[i]=' ';
17         i++;
18         vidmem[i]=WHITE_TXT;
19         i++;
20     };
21 }
22
23 /******
24 *setup_IDT_entry
25 * Inicializa un descriptor de la IDT
26 *
27 * Recibe: Puntero a elemento de la IDT
28 *         Selector a cargar en el descriptor de interrupcion
29 *         Puntero a rutina de atencion de interrupcion
30 *         Derechos de acceso del segmento
31 *         Cero
32 *****/
33
34 void setup_IDT_entry (DESCR_INT *item, byte selector, dword offset, ←
   byte access,
   byte cero) {
35     item->selector = selector;
36     item->offset_l = offset & 0xFFFF;
37     item->offset_h = offset >> 16;
38     item->access = access;
39     item->cero = cero;
40 }
41
```

1.9. loader.asm

driver

1.10. keyboard.c

11

```

    'O','P','[','\n','@','A','S','D','F','G','H','J','K','L','\n','←',
    '"','@','@','Z','X','C','V','B','N','M',';','/','@','@',
    ',',''};

23
24 int shift;
25 int bloq_mayusc;
26
27 int bloq_mayusc_unpresed();
28 int bloq_mayusc_presed();
29
30 int bloq_mayusc_presed(){
31     bloq_mayusc=0;
32     add_key_listener(-1,BLOQ_MAYUS_SCAN_CODE, bloq_mayusc_unpresed←
33         );
34     return 0;
35 }
36
37 int bloq_mayusc_unpresed(){
38     bloq_mayusc=1;
39     add_key_listener(-1,BLOQ_MAYUS_SCAN_CODE, bloq_mayusc_presed);
40     return 0;
41 }
42
43 int shift_presed(){
44     shift++;
45     actual_scan_code_table=SHIFT_SCAN_CODE_TABLE;
46     return 0;
47 }
48
49 int shift_released(){
50     shift--;
51     if(shift==0){
52         actual_scan_code_table=SCAN_CODE_TABLE;
53     }
54     return 0;
55 }
56
57 void IRQ1_handler(registers_t reg){
58     int tmp;
59     int i=inb(KEYBOARD);
60     if(activate(i)){
61         tmp=(stdin.end+1)%stdin.size;
62         if(tmp!=stdin.start){
63             char c=actual_scan_code_table[i];
64             if(bloq_mayusc){
65                 if(c>='a' && c<='z'){
66                     c=c+'A'-'a';
67                 } else if(c>='A' && c<='Z'){
68                     c=c+'a'-'A';
69                 }
70                 stdin.array[stdin.end]=c;
71                 stdin.end=tmp;
72             } else {
73                 //TODO: beep
74             }
75         }
76     }
77 }
78
79 static void reset(){
80     outb(0x64,0xFE);
81 }
82
83 static int cnrl_alt_supr_manager(){
84     reset();
85     return 0;
86 }
87
88 void init_keyboard(){
89     register_interrupt_handler(IRQ1, IRQ1_handler);
90     stdin.start=stdin.end=0;
91     stdin.array=array;
92     stdin.size=BUFFER_SIZE;

```

```

93     add_in_out(0,&stdin);
94     actual_scan_code_table=SCAN_CODE_TABLE;
95     bloq_mayusc=0;
96     init_key_listeners();
97     add_key_listener(-1,LSHIFT_KEY_PRESSED_SCAN_CODE, shift_pressed)↵
98     ;
99     add_key_listener(-1,RSHIFT_KEY_PRESSED_SCAN_CODE, shift_pressed)↵
100    ;
101    add_key_listener(-1,LSHIFT_KEY_RELEASED_SCAN_CODE, ↵
102    shift_released);
103    add_key_listener(-1,RSHIFT_KEY_RELEASED_SCAN_CODE, ↵
104    shift_released);
105    add_key_listener(-1,BLOQ_MAYUS_SCAN_CODE, bloq_mayusc_unpressed↵
106    );
107    add_key_listener(3, 83, cnrl_alt_supr_manager);
108 }

```

1.11. keyboard.h

```

1  #ifndef KEYBOARD_H
2  #define KEYBOARD_H
3
4  void init_keyboard();
5
6  #endif /* KEYBOARD_H */

```

1.12. screen.c

```

1  #include "screen.h"
2  #include "../system/isr.h"
3  #include "../system/in_out.h"
4  #include "timer.h"
5
6  int16_t *video_memory = (int16_t *) 0xB8000;
7
8  #define BUFFER_SIZE 1000
9  char array_out[BUFFER_SIZE];
10 buffer_t stdout;
11
12 #define ESC '\x1B'
13 #define BELL '\x07'
14
15 #define DEFAULT_SETTINGS 0x07
16
17 #define SCREEN_SIZE_X 80
18 #define SCREEN_SIZE_Y 25
19 uint8_t screen_state = 0; // 0=normal, 1=scaped, 2=parameters.
20
21 #define SCREEN_MAX_PARAM_COUNT 16
22 uint8_t screen_param_count = 0;
23 int screen_param[SCREEN_MAX_PARAM_COUNT];
24
25 uint8_t screen_cursor_x = 0;
26 uint8_t screen_cursor_y = 0;
27 uint8_t screen_settings = DEFAULT_SETTINGS;
28
29 #define VGA_HIGH_CURSOR_BYTE 14
30 #define VGA_LOW_CURSOR_BYTE 15
31 #define VGA_MODE_PORT 0x3D4
32 #define VGA_IO_PORT 0x3D5
33
34 static void update_cursor() {

```

```

35     int16_t cursorLocation = screen_cursor_y * SCREEN_SIZE_X + ↵
        screen_cursor_x;
36     outb(VGA_MODE_PORT, VGA_HIGH_CURSOR_BYTE);
37     outb(VGA_IO_PORT, cursorLocation >> 8);
38     outb(VGA_MODE_PORT, VGA_LOW_CURSOR_BYTE);
39     outb(VGA_IO_PORT, cursorLocation);
40 }
41
42 static void scroll() {
43     int16_t blank = ' ' | (DEFAULT_SETTINGS << 8);
44     if (screen_cursor_y >= SCREEN_SIZE_Y) {
45         int i;
46         for (i = 0 * SCREEN_SIZE_X; i < (SCREEN_SIZE_Y - 1) * ↵
            SCREEN_SIZE_X; i++) {
47             video_memory[i] = video_memory[i + SCREEN_SIZE_X];
48         }
49         int lastLine = SCREEN_SIZE_Y - 1;
50         for (i = (lastLine) * SCREEN_SIZE_X; i < SCREEN_SIZE_Y * ↵
            SCREEN_SIZE_X; i++) {
51             video_memory[i] = blank;
52         }
53         screen_cursor_y = (lastLine);
54     }
55 }
56
57 static void print(char c) {
58     int16_t *location;
59     location = video_memory + (screen_cursor_y * SCREEN_SIZE_X + ↵
        screen_cursor_x);
60
61     if (c != '\b') {
62         *location = (c | (screen_settings << 8));
63         if (++screen_cursor_x >= SCREEN_SIZE_X) {
64             screen_cursor_x = 0;
65             screen_cursor_y++;
66         }
67     } else {
68         *location = (' ' | (screen_settings << 8));
69     }
70 }
71
72 static void do_bell() {
73     // TODO
74 }
75
76 static void do_backspace() {
77     if (screen_cursor_x) {
78         screen_cursor_x--;
79     } else if (screen_cursor_y) {
80         screen_cursor_x = SCREEN_SIZE_X - 1;
81         screen_cursor_y--;
82     }
83     print('\b');
84 }
85
86 static void do_lineFeed() {
87     screen_cursor_x = 0;
88     screen_cursor_y++;
89 }
90
91 static void do_tab() {
92     screen_cursor_x = (screen_cursor_x + 4) & ~(4 - 1);
93 }
94
95 static void do_return() {
96     screen_cursor_x = 0;
97 }
98
99 static void screen_clear() {
100     int16_t blank = ' ' | (DEFAULT_SETTINGS << 8);
101     int i;
102     for (i = 0; i < SCREEN_SIZE_X * SCREEN_SIZE_Y; i++) {
103         video_memory[i] = blank;
104     }

```

```

105     screen_cursor_x = screen_cursor_y = 0;
106     update_cursor();
107 }
108
109 static void do_scape_J() {
110     if (screen_param[0] == 2) {
111         screen_clear();
112     }
113 }
114
115 /* Map from ANSI colors to the attributes used by the PC */
116 static uint8_t ansi_colors[8] = {0, 4, 2, 6, 1, 5, 3, 7};
117
118 static void do_scape_m() {
119     int i;
120     for (i = 0; i < screen_param_count; i++) {
121         int dec = screen_param[i] / 10;
122         int u = screen_param[i] % 10;
123         if (dec == 0) {
124             switch (u) {
125                 case 0:
126                     screen_settings = DEFAULT_SETTINGS;
127                     break;
128                 case 1:
129                     screen_settings |= 0x08;
130                     break;
131                 case 4:
132                     screen_settings &= 0xBB;
133                     break;
134                 case 5:
135                     screen_settings |= 0x80;
136             }
137         } else if (dec == 3) { /* foreground */
138             //print('3');
139             screen_settings = (0xF0 & screen_settings) | (0x0F &
140                 ansi_colors[u]);
141         } else if (dec == 4) { /* background */
142             screen_settings = (0x0F & screen_settings) | (ansi_colors[
143                 u] << 4);
144         }
145     }
146 }
147
148 static void do_scape(char c) {
149     switch (screen_state) {
150         case 1:
151             if (c == '[') {
152                 screen_state = 2;
153                 screen_param_count = 1;
154                 int i = 0;
155                 for (; i <= SCREEN_MAX_PARAM_COUNT; i++) {
156                     screen_param[i] = 0;
157                 }
158             } else {
159                 screen_state = 0;
160             }
161             break;
162         case 2:
163             if (c >= '0' && c <= '9') {
164                 screen_param[screen_param_count - 1] = 10 *
165                     screen_param[screen_param_count - 1] + (c - '0');
166             } else if (c == ';') {
167                 screen_param_count++;
168             } else {
169                 switch (c) {
170                     case 'm':
171                         do_scape_m();
172                         break;
173                     case 'J':
174                         do_scape_J();
175                         break;
176                 }
177                 screen_state = 0;
178             }
179     }
180 }

```

```
176         break;
177     }
178 }
179
180 void screen_put(char c) {
181     if (screen_state > 0) {
182         do_scape(c);
183         return;
184     } else {
185         switch (c) {
186             case ESC:
187                 screen_state = 1;
188                 return;
189             case '\0':
190                 return;
191             case BELL:
192                 do_bell();
193                 return;
194             case '\b':
195                 do_backspace();
196                 break;
197             case '\n':
198                 do_lineFeed();
199                 break;
200             case '\t':
201                 do_tab();
202                 break;
203             case '\r':
204                 do_return();
205                 break;
206             default:
207                 print(c);
208                 break;
209         }
210         scroll();
211         update_cursor();
212     }
213 }
214
215 void screen_write(char *string) {
216     int i = 0;
217     while (string[i]) {
218         screen_put(string[i++]);
219     }
220 }
221
222 static void timer_print(registers_t reg) {
223     int i;
224     for (i = 0; stdout.start != stdout.end; i++) {
225         screen_put(stdout.array[stdout.start]);
226         stdout.start = (stdout.start + 1) % stdout.size;
227     }
228 }
229
230 void init_screen() {
231     register_tick_subhandler(timer_print);
232     stdout.start = stdout.end = 0;
233     stdout.array = array_out;
234     stdout.size = BUFFER_SIZE;
235     add_in_out(1, &stdout);
236     screen_write("\x1B[2J");
237 }
```

1.13. screen.h

```
1  /**
2  * screen.h | Interfaz para manejo de pantalla.
3  */
4  #include "../system/common.h"
```



```

5
6 #ifndef SCREEN_H
7 #define SCREEN_H
8 /**
9  * Escribe un caracter en pantalla.
10  * @param char c: el caracter a escribir.
11  * Los siguientes ANSI scape Characters fueron implementados:
12  *
13  *      Esc[2J          Borra la pantalla y mueve el cursor a (line 0, ←
14  *          column 0).
15  *      Esc[##;##;...m  Cambia el modo de graficos segun los ←
16  *          siguientes atributos:
17  *
18  *      Text attributes
19  *      0      All attributes off
20  *      1      Bold on
21  *      4      Underscore (on monochrome display adapter only)
22  *      5      Blink on
23  *
24  *      Foreground colors      Background colors
25  *      30      Black          40      Black
26  *      31      Red            41      Red
27  *      32      Green          42      Green
28  *      33      Yellow         43      Yellow
29  *      34      Blue           44      Blue
30  *      35      Magenta        45      Magenta
31  *      36      Cyan           46      Cyan
32  *      37      White          47      White
33  *
34  * Ej: Esc[34;47m (azul en fondo blanco)
35  */
36 void screen_put(char c);
37 #endif

```

1.14. timer.c

```

1 #include "../system/isr.h"
2 #include "../system/int80.h"
3
4 #define SUB_FUNC_VEC_SIZE 10
5
6 int80_t sub_handler_vec[SUB_FUNC_VEC_SIZE];
7
8 int ticks;
9 int count_ticks;
10 int sub_func_count;
11 unsigned long k;
12
13 void register_tick_subhandler(int80_t func) {
14     if (sub_func_count < SUB_FUNC_VEC_SIZE - 1) {
15         sub_handler_vec[sub_func_count] = func;
16         sub_func_count++;
17     }
18 }
19
20 void IRQ0_handler(registers_t regs) {
21     int i;
22     if (count_ticks) {
23         if (ticks == 0) {
24             k = getRDTSC();
25         }
26         ticks++;
27     }
28     for (i = 0; i < sub_func_count; i++) {
29         sub_handler_vec[i](regs);
30     }
31 }

```

```

32
33 void cpu_speed(registers_t regs) {
34     count_ticks = 1;
35     ticks = -1;
36     _Sti();
37     while (ticks < 30);
38     k = getRDTSC() - k;
39     _Cli();
40     count_ticks = 0;
41     *((unsigned long*) regs.ebx) = (k / ticks)*18 + k / (ticks * 5);
42 }
43
44 void init_timer_tick() {
45     sub_func_count = 0;
46     count_ticks = 0;
47     register_interrupt_handler(IRQ0, IRQ0_handler);
48     register_functionality(5, cpu_speed);
49 }

```

1.15. timer.h

```

1 #include "../system/int80.h"
2
3 #ifndef TIMER_H
4 #define TIMER_H
5
6 void register_tick_subhandler(int80_t func);
7
8 void init_timer_tick();
9
10
11 void start_ticks();
12 void stop_ticks();
13 int get_ticks();
14 #endif /* TIMER_H */

```

system

1.16. common.h

```

1 #ifndef COMMON_H
2 #define COMMON_H
3
4 // Exact-width integer types
5 typedef signed char int8_t;
6 typedef unsigned char uint8_t;
7 typedef signed short int16_t;
8 typedef unsigned short uint16_t;
9 typedef signed int int32_t;
10 typedef unsigned int uint32_t;
11
12 #define NULL ((void*)0)
13
14 // PIC
15 #define PORT_PIC1 0x20
16 #define PORT_PIC2 0xA0
17 #define SIGNAL_EOI 0x20
18
19 extern void outw(uint16_t port, uint16_t value);
20 extern void outb(uint16_t port, uint8_t value);
21 extern uint8_t inb(uint16_t port);
22 extern uint16_t inw(uint16_t port);
23 extern uint32_t getRDTSC();

```

```

24 |
25 | #endif // COMMON_H

```

1.17. idt.c

```

1  #include "common.h"
2  #include "idt.h"
3  #include "isr.h"
4
5  static void init_idt();
6  static void idt_set_gate(uint8_t, uint32_t, uint16_t, uint8_t);
7
8  idt_entry_t idt_entries[256];
9  idt_ptr_t    idt_ptr;
10
11 // Extern the ISR handler array so we can nullify them on startup.
12 extern isr_t interrupt_handlers[];
13 extern void idt_flush(uint32_t);
14
15 void init_descriptor_tables()
16 {
17     /* Habilito interrupcion de timer tick*/
18     _Cli();
19     _mascaraPIC1(0xFE);
20     _mascaraPIC2(0xFF);
21     _Sti();
22
23     init_idt();
24 }
25
26
27 static void init_idt()
28 {
29     idt_ptr.limit = sizeof(idt_entry_t) * 256 - 1;
30     idt_ptr.base  = (uint32_t)&idt_entries;
31
32     // Remap the irq table.
33     outb(0x20, 0x11);
34     outb(0xA0, 0x11);
35     outb(0x21, 0x20);
36     outb(0xA1, 0x28);
37     outb(0x21, 0x04);
38     outb(0xA1, 0x02);
39     outb(0x21, 0x01);
40     outb(0xA1, 0x01);
41     outb(0x21, 0x00);
42     outb(0xA1, 0x00);
43
44     idt_set_gate(0, (uint32_t)isr0, 0x08, 0x8E);
45     idt_set_gate(1, (uint32_t)isr1, 0x08, 0x8E);
46     idt_set_gate(2, (uint32_t)isr2, 0x08, 0x8E);
47     idt_set_gate(3, (uint32_t)isr3, 0x08, 0x8E);
48     idt_set_gate(4, (uint32_t)isr4, 0x08, 0x8E);
49     idt_set_gate(5, (uint32_t)isr5, 0x08, 0x8E);
50     idt_set_gate(6, (uint32_t)isr6, 0x08, 0x8E);
51     idt_set_gate(7, (uint32_t)isr7, 0x08, 0x8E);
52     idt_set_gate(8, (uint32_t)isr8, 0x08, 0x8E);
53     idt_set_gate(9, (uint32_t)isr9, 0x08, 0x8E);
54     idt_set_gate(10, (uint32_t)isr10, 0x08, 0x8E);
55     idt_set_gate(11, (uint32_t)isr11, 0x08, 0x8E);
56     idt_set_gate(12, (uint32_t)isr12, 0x08, 0x8E);
57     idt_set_gate(13, (uint32_t)isr13, 0x08, 0x8E);
58     idt_set_gate(14, (uint32_t)isr14, 0x08, 0x8E);
59     idt_set_gate(15, (uint32_t)isr15, 0x08, 0x8E);
60     idt_set_gate(16, (uint32_t)isr16, 0x08, 0x8E);
61     idt_set_gate(17, (uint32_t)isr17, 0x08, 0x8E);
62     idt_set_gate(18, (uint32_t)isr18, 0x08, 0x8E);
63     idt_set_gate(19, (uint32_t)isr19, 0x08, 0x8E);
64     idt_set_gate(20, (uint32_t)isr20, 0x08, 0x8E);

```

```

65     idt_set_gate(21, (uint32_t) isr21, 0x08, 0x8E);
66     idt_set_gate(22, (uint32_t) isr22, 0x08, 0x8E);
67     idt_set_gate(23, (uint32_t) isr23, 0x08, 0x8E);
68     idt_set_gate(24, (uint32_t) isr24, 0x08, 0x8E);
69     idt_set_gate(25, (uint32_t) isr25, 0x08, 0x8E);
70     idt_set_gate(26, (uint32_t) isr26, 0x08, 0x8E);
71     idt_set_gate(27, (uint32_t) isr27, 0x08, 0x8E);
72     idt_set_gate(28, (uint32_t) isr28, 0x08, 0x8E);
73     idt_set_gate(29, (uint32_t) isr29, 0x08, 0x8E);
74     idt_set_gate(30, (uint32_t) isr30, 0x08, 0x8E);
75     idt_set_gate(31, (uint32_t) isr31, 0x08, 0x8E);
76
77     idt_set_gate(32, (uint32_t) irq0, 0x08, 0x8E);
78     idt_set_gate(33, (uint32_t) irq1, 0x08, 0x8E);
79     idt_set_gate(34, (uint32_t) irq2, 0x08, 0x8E);
80     idt_set_gate(35, (uint32_t) irq3, 0x08, 0x8E);
81     idt_set_gate(36, (uint32_t) irq4, 0x08, 0x8E);
82     idt_set_gate(37, (uint32_t) irq5, 0x08, 0x8E);
83     idt_set_gate(38, (uint32_t) irq6, 0x08, 0x8E);
84     idt_set_gate(39, (uint32_t) irq7, 0x08, 0x8E);
85     idt_set_gate(40, (uint32_t) irq8, 0x08, 0x8E);
86     idt_set_gate(41, (uint32_t) irq9, 0x08, 0x8E);
87     idt_set_gate(42, (uint32_t) irq10, 0x08, 0x8E);
88     idt_set_gate(43, (uint32_t) irq11, 0x08, 0x8E);
89     idt_set_gate(44, (uint32_t) irq12, 0x08, 0x8E);
90     idt_set_gate(45, (uint32_t) irq13, 0x08, 0x8E);
91     idt_set_gate(46, (uint32_t) irq14, 0x08, 0x8E);
92     idt_set_gate(47, (uint32_t) irq15, 0x08, 0x8E);
93
94
95     idt_set_gate(0x80, (uint32_t) isr80h, 0x08, 0x8E);
96
97
98     idt_flush((uint32_t)&idt_ptr);
99 }
100
101 static void idt_set_gate(uint8_t num, uint32_t base, uint16_t sel, ←
    uint8_t flags)
102 {
103     idt_entries[num].base_lo = base & 0xFFFF;
104     idt_entries[num].base_hi = (base >> 16) & 0xFFFF;
105
106     idt_entries[num].sel      = sel;
107     idt_entries[num].always0 = 0;
108
109     idt_entries[num].flags    = flags;
110 }

```

1.18. idt.h

```

1  #include "common.h"
2
3  void init_descriptor_tables();
4
5  // interrupt gate descriptor
6  struct idt_entry_struct {
7      uint16_t base_lo;
8      uint16_t sel;
9      uint8_t always0;
10     uint8_t flags;
11     uint16_t base_hi;
12 } __attribute__((packed));
13
14 typedef struct idt_entry_struct idt_entry_t;
15
16 // array of interrupt handlers descriptor (for lidt).
17 struct idt_ptr_struct {
18     uint16_t limit;
19     uint32_t base;

```

```
20 } __attribute__((packed));
21
22 typedef struct idt_ptr_struct idt_ptr_t;
23
24 #define IDT_SIZE 256
25
26 // interrupciones default del procesador.
27 extern void isr0();
28 extern void isr1();
29 extern void isr2();
30 extern void isr3();
31 extern void isr4();
32 extern void isr5();
33 extern void isr6();
34 extern void isr7();
35 extern void isr8();
36 extern void isr9();
37 extern void isr10();
38 extern void isr11();
39 extern void isr12();
40 extern void isr13();
41 extern void isr14();
42 extern void isr15();
43 extern void isr16();
44 extern void isr17();
45 extern void isr18();
46 extern void isr19();
47 extern void isr20();
48 extern void isr21();
49 extern void isr22();
50 extern void isr23();
51 extern void isr24();
52 extern void isr25();
53 extern void isr26();
54 extern void isr27();
55 extern void isr28();
56 extern void isr29();
57 extern void isr30();
58 extern void isr31();
59 extern void irq0();
60 extern void irq1();
61 extern void irq2();
62 extern void irq3();
63 extern void irq4();
64 extern void irq5();
65 extern void irq6();
66 extern void irq7();
67 extern void irq8();
68 extern void irq9();
69 extern void irq10();
70 extern void irq11();
71 extern void irq12();
72 extern void irq13();
73 extern void irq14();
74 extern void irq15();
75
76 extern void isr80h();
```

1.19. in_out.c

```
1 #include "int80.h"
2 #include "in_out.h"
3
4 buffer_t * in_out_vector[10];
5
6 void READ_INTERRUPT_handler(registers_t regs){
7     int i;
8     buffer_t * buff=in_out_vector[regs.ebx];
9     for(i=0;i<regs.edx && buff->start!=buff->end;i++){
```

```

10         *((char*)(regs.ecx+i))=buff->array[buff->start];
11         buff->start=(buff->start+1)%buff->size;
12     }
13     if(i<regs.edx){
14         *((char*)(regs.ecx+i))='\0';
15     }
16 }
17
18 void WRITE_INTERRUPT_handler(registers_t regs){
19     int i;
20     int tmp;
21     buffer_t * buff=in_out_vector[regs.ebx];
22     tmp=(buff->end+1)%buff->size;
23     for(i=0;i<regs.edx && tmp!=buff->start;i++,tmp=(buff->end+1)%buff->size){
24         buff->array[buff->end]=*((char*)(regs.ecx+i));
25         buff->end=tmp;
26     }
27 }
28
29 void add_in_out(int n, buffer_t * buff){
30     in_out_vector[n]=buff;
31 }
32
33
34 init_in_out(){
35     register_functionality(3,READ_INTERRUPT_handler);
36     register_functionality(4,WRITE_INTERRUPT_handler);
37 }

```

1.20. in_out.h

```

1 #ifndef IN_H
2 #define IN_H
3
4
5 struct buffer_struct
6 {
7     int size;
8     char * array;
9     int start;
10    int end;
11 };
12
13 typedef struct buffer_struct buffer_t;
14
15 #endif // IN_OUT_H

```

1.21. int80.c

```

1 #include "isr.h"
2 #include "int80.h"
3
4 #define SUB_FUNC_VEC_SIZE 10
5
6
7
8 int80_t sub_funcs_vec[SUB_FUNC_VEC_SIZE];
9
10
11 void register_functionality(uint8_t n, int80_t func) {
12     if(n<SUB_FUNC_VEC_SIZE){
13         sub_funcs_vec[n] = func;
14     }
15 }

```

```
14     }
15 }
16
17 void int80_handler(registers_t regs){
18     if(regs.eax<SUB_FUNC_VEC_SIZE){
19         sub_funcs_vec[regs.eax](regs);
20     }
21 }
22
23 void nofunc(registers_t regs){
24 }
25
26
27
28 void init_int80(){
29     int i;
30     for(i=0;i<SUB_FUNC_VEC_SIZE;i++){
31         sub_funcs_vec[i]=nofunc;
32     }
33     register_interrupt_handler(0x80,int80_handler);
34 }
```

1.22. int80.h

```
1 #include "isr.h"
2
3 #ifndef INT80_H
4 #define INT80_H
5
6 typedef void (*int80_t)(registers_t);
7 void register_functionality(uint8_t n, int80_t func);
8 void init_int80();
9
10 #endif /* INT80_H */
```

1.23. isr.c

```
1 #include "common.h"
2 #include "isr.h"
3 #include "idt.h"
4
5 isr_t interrupt_handlers[IDT_SIZE];
6
7 void register_interrupt_handler(uint8_t n, isr_t handler) {
8     interrupt_handlers[n] = handler;
9 }
10
11 void isr_handler(registers_t regs) {
12     if(regs.int_no==128){ //cableo orrendo, pero por alguna razon me ←
13         lo pone negativo
14         regs.int_no*=-1;
15     }
16     if (interrupt_handlers[regs.int_no] != NULL) {
17         isr_t handler = interrupt_handlers[regs.int_no];
18         handler(regs);
19     }
20 }
21
22 void irq_handler(registers_t regs) {
23     if (regs.int_no >= IRQ8) {
24         outb(PORT_PIC2, SIGNAL_EOI);
25     }
26     outb(PORT_PIC1, SIGNAL_EOI);
27 }
```

```
26     isr_handler(regs);
27 }
```

1.24. isr.h

```
1  #include "common.h"
2
3  #ifndef ISR_H
4  #define ISR_H
5
6  #define IRQ0 32
7  #define IRQ1 33
8  #define IRQ2 34
9  #define IRQ3 35
10 #define IRQ4 36
11 #define IRQ5 37
12 #define IRQ6 38
13 #define IRQ7 39
14 #define IRQ8 40
15 #define IRQ9 41
16 #define IRQ10 42
17 #define IRQ11 43
18 #define IRQ12 44
19 #define IRQ13 45
20 #define IRQ14 46
21 #define IRQ15 47
22
23 typedef struct registers
24 {
25     uint32_t ds;
26     uint32_t edi, esi, ebp, esp, ebx, edx, ecx, eax; // pusha pushs.
27     uint32_t int_no, err_code;
28     uint32_t eip, cs, eflags, useresp, ss; // processor automatic ←
29     // pushs.
30 } registers_t;
31
32 typedef void (*isr_t)(registers_t);
33 void register_interrupt_handler(uint8_t n, isr_t handler);
34 #endif //ISR_H
```

1.25. keyboardlisteners.c

```
1  #ifndef KEYBOARDLISTENER_H
2  #define KEYBOARDLISTENER_H
3
4  #define MAX_SCAN_CODE 300
5
6  #define CTRL_KEY_PRESSED_SCAN_CODE 29
7  #define CTRL_KEY_RELEASED_SCAN_CODE 157
8
9  #define ALT_KEY_PRESSED_SCAN_CODE 56
10 #define ALT_KEY_RELEASED_SCAN_CODE 184
11
12 typedef int (*key_listener)();
13
14 int activate(int scan_code);
15 void add_key_listener(int mode, int scan_code, key_listener listener);
16 void init_key_listeners();
17
18 #endif //KEYBOARDLISTENER_H
```


1.26. keyboardlisteners.h

```

1  #ifndef KEYBOARDLISTENER_H
2  #define KEYBOARDLISTENER_H
3
4  #define MAX_SCAN_CODE 300
5
6  #define CTRL_KEY_PRESSED_SCAN_CODE 29
7  #define CTRL_KEY_RELEASED_SCAN_CODE 157
8
9  #define ALT_KEY_PRESSED_SCAN_CODE 56
10 #define ALT_KEY_RELEASED_SCAN_CODE 184
11
12 typedef int (*key_listener)();
13
14 int activate(int scan_code);
15 void add_key_listener(int mode, int scan_code, key_listener listener);
16 void init_key_listeners();
17
18 #endif //KEYBOARDLISTENER_H

```

asm

1.27. idt.asm

```

1  [GLOBAL idt_flush] ; Allows the C code to call idt_flush().
2
3  idt_flush:
4      mov eax, [esp+4] ; Get the pointer to the IDT, passed as a ↵
5      parameter.
6      lidt [eax] ; Load the IDT pointer.
7      ret
8
9  %macro ISR_NOERRCODE 1
10     global isr %1
11     isr %1:
12         cli ; Disable interrupts firstly.
13         push byte 0 ; Push a dummy error code.
14         push byte %1 ; Push the interrupt number.
15         jmp isr_common_stub ; Go to our common handler code.
16 %endmacro
17
18 ; This macro creates a stub for an ISR which passes it's own
19 ; error code.
20 %macro ISR_ERRCODE 1
21     global isr %1
22     isr %1:
23         cli ; Disable interrupts.
24         push byte %1 ; Push the interrupt number
25         jmp isr_common_stub
26 %endmacro
27
28 ; This macro creates a stub for an IRQ — the first parameter is
29 ; the IRQ number, the second is the ISR number it is remapped to.
30 %macro IRQ 2
31     global irq %1
32     irq %1:
33         cli
34         push byte 0
35         push byte %2
36         jmp irq_common_stub
37 %endmacro
38
39 ISR_NOERRCODE 0
40 ISR_NOERRCODE 1
41 ISR_NOERRCODE 2

```

```

41 | ISR_NOERRCODE 3
42 | ISR_NOERRCODE 4
43 | ISR_NOERRCODE 5
44 | ISR_NOERRCODE 6
45 | ISR_NOERRCODE 7
46 | ISR_ERRCODE 8
47 | ISR_NOERRCODE 9
48 | ISR_ERRCODE 10
49 | ISR_ERRCODE 11
50 | ISR_ERRCODE 12
51 | ISR_ERRCODE 13
52 | ISR_ERRCODE 14
53 | ISR_NOERRCODE 15
54 | ISR_NOERRCODE 16
55 | ISR_NOERRCODE 17
56 | ISR_NOERRCODE 18
57 | ISR_NOERRCODE 19
58 | ISR_NOERRCODE 20
59 | ISR_NOERRCODE 21
60 | ISR_NOERRCODE 22
61 | ISR_NOERRCODE 23
62 | ISR_NOERRCODE 24
63 | ISR_NOERRCODE 25
64 | ISR_NOERRCODE 26
65 | ISR_NOERRCODE 27
66 | ISR_NOERRCODE 28
67 | ISR_NOERRCODE 29
68 | ISR_NOERRCODE 30
69 | ISR_NOERRCODE 31
70 |
71 | IRQ 0, 32
72 | IRQ 1, 33
73 | IRQ 2, 34
74 | IRQ 3, 35
75 | IRQ 4, 36
76 | IRQ 5, 37
77 | IRQ 6, 38
78 | IRQ 7, 39
79 | IRQ 8, 40
80 | IRQ 9, 41
81 | IRQ 10, 42
82 | IRQ 11, 43
83 | IRQ 12, 44
84 | IRQ 13, 45
85 | IRQ 14, 46
86 | IRQ 15, 47
87 |
88 | global isr80h
89 | isr80h:
90 | cli ; Disable interrupts firstly.
91 | push byte 0 ; Push a dummy error code.
92 | push byte 80h ; Push the interrupt number.
93 | jmp isr_common_stub ; Go to our common handler code.
94 |
95 |
96 | ; In isr.c
97 | extern isr_handler
98 |
99 | ; ISR stub. It saves the processor state, sets
100 | ; up for kernel mode segments, calls the C-level fault handler,
101 | ; and finally restores the stack frame.
102 | isr_common_stub:
103 | pusha ; Pushes edi,esi,ebp,esp,ebx,edx,ecx,eax
104 |
105 | mov ax, ds ; Lower 16-bits of eax = ds.
106 | push eax ; save the data segment descriptor
107 |
108 | mov ax, 0x10 ; load the kernel data segment descriptor
109 | mov ds, ax
110 | mov es, ax
111 | mov fs, ax
112 | mov gs, ax
113 |
114 | call isr_handler

```

```

115
116     pop ebx          ; reload the original data segment descriptor
117     mov ds, bx
118     mov es, bx
119     mov fs, bx
120     mov gs, bx
121
122     popa              ; Pops edi,esi,ebp,...
123     add esp, 8        ; Cleans up the pushed error code and pushed ISR ←
                        number
124     sti
125     iret              ; pops 5 things at once: CS, EIP, EFLAGS, SS, and ←
                        ESP
126
127 ; In isr.c
128 extern irq_handler
129
130 ; IRQ stub. It saves the processor state, sets
131 ; up for kernel mode segments, calls the C-level fault handler,
132 ; and finally restores the stack frame.
133 irq_common_stub:
134     pusha              ; Pushes edi,esi,ebp,esp,ebx,edx,ecx,eax
135
136     mov ax, ds         ; Lower 16-bits of eax = ds.
137     push eax           ; save the data segment descriptor
138
139     mov ax, 0x10       ; load the kernel data segment descriptor
140     mov ds, ax
141     mov es, ax
142     mov fs, ax
143     mov gs, ax
144
145     call irq_handler
146
147     pop ebx          ; reload the original data segment descriptor
148     mov ds, bx
149     mov es, bx
150     mov fs, bx
151     mov gs, bx
152
153     popa              ; Pops edi,esi,ebp,...
154     add esp, 8        ; Cleans up the pushed error code and pushed ISR ←
                        number
155     sti
156     iret              ; pops 5 things at once: CS, EIP, EFLAGS, SS, and ←
                        ESP

```

1.28. common.asm

```

1  global outb
2  global outw
3  global inb
4  global inw
5  global getRDTSC
6
7  getRDTSC:
8      rdtsc
9      ret
10
11 outb:
12     mov dx, [esp+4]
13     mov al, [esp+8]
14     out dx, al
15     ret
16
17 outw:
18     mov dx, [esp+4]
19     mov ax, [esp+8]
20     out dx, ax

```

```
21     ret
22
23 inb:
24     mov dx, [esp+4]
25     in al, dx
26     ret
27
28 inw:
29     mov dx, [esp+4]
30     in ax, dx
31     ret
```

std

1.29. getchar.c

```
1  #include "stdio.h"
2
3  #define STREAM_SIZE 500
4
5  typedef int (*flusher)(char * streampointer);
6
7  char stream[STREAM_SIZE];
8  char * streamout=stream;
9
10
11 int intro_flush(char * streampointer){
12     if(*streampointer=='\n' || 1>=STREAM_SIZE-(streampointer-stream)←
13         -1){
14         return 1;
15     }
16     return 0;
17 }
18
19 char getchar(){
20     char c=*streamout;
21     if(c=='\0'){
22         streamout=stream;
23         char * streamin=stream;
24         int i,j;
25         for(i=0;i<STREAM_SIZE;i++){
26             stream[i]='\0';
27         }
28         while(!intro_flush(streamin)){
29             if(*streamin!='\0')
30                 streamin++;
31             __read(0,streamin,1);
32             if(*streamin!='\b' && *streamin!='\t'){
33                 printf(streamin);
34             }
35             else if(*streamin=='\b'){
36                 if(streamin > stream){
37                     printf("\b");
38                     *streamin='\0';
39                     streamin--;
40                 }
41                 *streamin='\0';
42             } else if(*streamin=='\t'){
43                 *streamin='\0';
44             }
45         }
46         c=*streamout;
47     }
48     streamout++;
49     return c;
50 }
51 }
```

1.30. printf.c

```
1 #include "stdio.h"
2
3 static void prints(char * string);
4
5 static char * numberBaseNtoString(unsigned int number, int base, char ↵
    * out);
6
7 void putchar(char c) {
8     _write(1, &c, 1);
9 }
10
11 void printf(char * formatString, ...) {
12     int integer;
13     unsigned int unsignedInteger;
14     char * string;
15     char out[40];
16     char c;
17
18     va_list args;
19
20     va_start(args, formatString);
21
22     while (*formatString != '\0') {
23         if (*formatString == '%') {
24
25             formatString++;
26
27             switch (*formatString) {
28                 case 'c':
29                     c = va_arg(args, int);
30                     putchar(c);
31                     break;
32                 case 's':
33                     string = va_arg(args, char *);
34                     prints(string);
35                     break;
36                 case 'd':
37                     integer = va_arg(args, int);
38                     if (integer < 0) {
39                         integer = -integer;
40                         putchar('-');
41                     }
42                     prints(numberBaseNtoString(integer, 10, out));
43                     break;
44                 case 'u':
45                     unsignedInteger = va_arg(args, unsigned int);
46                     prints(numberBaseNtoString(unsignedInteger, 10, ↵
                        out));
47                     break;
48                 case 'o':
49                     integer = va_arg(args, unsigned int);
50                     prints(numberBaseNtoString(integer, 8, out));
51                     break;
52                 case 'x':
53                     unsignedInteger = va_arg(args, unsigned int);
54                     prints(numberBaseNtoString(unsignedInteger, 16, ↵
                        out));
55                     break;
56                 case '%':
57                     putchar('%');
58                     break;
59             }
60             } else {
61                 putchar(*formatString);
62             }
63             formatString++;
64         }
65         va_end(args);
66     }
```

```
67
68 static void prints(char * string) {
69     while (*string != '\0') {
70         putchar(*string);
71         string++;
72     }
73 }
74
75 static char * numberBaseNtoString(unsigned int number, int base, char * out) {
76
77     int digits[40];
78     int position = 0;
79     char * numbers = "0123456789ABCDEF";
80     int index = 0;
81
82     if (number != 0) {
83         while (number > 0) {
84             if (number < base) {
85                 digits[position] = number;
86                 number = 0;
87             } else {
88                 digits[position] = number % base;
89                 number /= base;
90             }
91             position++;
92         }
93
94         for (index = 0; position > 0; position--, index++) {
95             out[index] = numbers[digits[position - 1] % base];
96         }
97         out[index] = '\0';
98     } else {
99         out[0] = '0';
100        out[1] = '\0';
101    }
102
103    return out;
104 }
```

1.31. scanf.c

```
1 #include "../src/std/string.h"
2 #include "stdio.h"
3
4 static int isNumber(char c) {
5     return (c >= '0' && c <= '9');
6 }
7
8 int sscanf(char *stream, char *format, ...) {
9     va_list ap;
10    va_start(ap, format);
11    int i = 0;
12    int j = 0;
13    int converted;
14
15    int *integer, iTmp, iTmp2;
16    char* string;
17    char *chr;
18    unsigned int *uinteger;
19
20    while (format[i]) {
21        if (format[i] == '%') {
22            i++;
23            switch (format[i++]) {
24                case 'c':
25                    chr = va_arg(ap, char*);
26                    *chr = stream[j++];
27                    break;
```

```

28         case 'd':
29             integer = va_arg(ap, int *);
30             iTmp = 0;
31             iTmp2 = 1;
32             if (stream[j] == '-') {
33                 iTmp2 = -1;
34                 j++;
35             }
36             while (isNumber(stream[j])) {
37                 iTmp = iTmp * 10 + (stream[j] - '0');
38                 j++;
39             }
40             *integer = iTmp*iTmp2;
41         case 'u':
42             uinteger = va_arg(ap, unsigned int *);
43             iTmp = 0;
44             while (isNumber(stream[j])) {
45                 iTmp = iTmp * 10 + (stream[j] - '0');
46                 j++;
47             }
48             *uinteger = iTmp;
49             break;
50         case 's':
51             string = va_arg(ap, char *);
52             iTmp = 0;
53             while (stream[j] != '\0') {
54                 string[iTmp++] = stream[j++];
55             }
56             string[iTmp] = '\0';
57             break;
58         default:
59             // WRONG %X
60             return converted;
61     }
62     } else {
63         if (format[i] == stream[j]) {
64             i++;
65             j++;
66         } else {
67             //WRONG FORMAT STRING
68             return converted;
69         }
70     }
71 }
72 }

```

1.32. stdio.h

```

1  #include "../include/varargs.h"
2  #include "../include/stdarg.h"
3
4  #ifndef STDIO_H
5  #define STDIO_H
6
7  char getchar();
8  void putchar(char c);
9  void printf(char *formatString, ...);
10 int sscanf(char *formatString, char *format, ...);
11
12 #endif //STDIO_H

```

1.33. string.c

```
1
2 int strcmp(char* str1, char * str2) {
3     int i;
4     for (i = 0; str1[i] != '\0' && str1[i] != '\0'; i++) {
5         if (str1[i] != str2[i]) {
6             return str1[i] - str2[i];
7         }
8     }
9     if (str1[i] == '\0' && str2[i] == '\0') {
10        return str1[i] - str2[i];
11    }
12    return 1;
13 }
14
15 void strcpy(char * str_des, char * str_ori) {
16     int i;
17     for (i = 0; str_ori[i] != '\0'; i++) {
18         str_des[i] = str_ori[i];
19     }
20     str_des[i] = '\0';
21 }
22
23 void strncpy(char * str_des, char * str_ori, unsigned int count) {
24     int i;
25     for (i = 0; str_ori[i] != '\0' && i <= count; i++) {
26         str_des[i] = str_ori[i];
27     }
28     str_des[i] = '\0';
29 }
30
31 int strlen(char* str) {
32     int i;
33     for (i = 0; str[i] != '\0'; i++);
34     return i;
35 }
```

1.34. string.h

```
1 #ifndef STRING_H
2 #define STRING_H
3
4 int strcmp(char* str1, char * str2);
5 void strcpy(char * str_des, char * str_ori);
6 int strlen(char* str);
7
8 #endif /* STRING_H */
```

1.35. syscall.asm

```
1 global __read
2 global __write
3 global __cpuspeed
4
5 SECTION .text
6
7 __read:
8     mov ecx, [esp+8]
9     mov eax, 3
10    mov ebx, [esp+4]
11    mov edx, [esp+12]
12    int 80h
13    ret
14
```



```
15 __write:
16     mov ecx, [esp+8]
17     mov eax, 4
18     mov ebx, [esp+4]
19     mov edx, [esp+12]
20     int 80h
21     ret
22
23 __cpuspeed:
24     mov ebx, [esp+4]
25     mov eax, 5
26     int 80h
27     ret
```

1.36. syscall.h

```
1 #ifndef SYSTEMCALL_H
2 #define SYSTEMCALL_H
3
4 void __read(int fd, void* buffer, int count);
5 void __write(int fd, const void* buffer, int count);
6 void __cpuspeed(void * ips);
7
8 #endif /* SYSTEMCALL_H */
```

user

1.37. commands.c

```
1 #include "commands.h"
2
3 #include "../std/string.h"
4
5 #define NULL 0
6 #define COMMAND_MAX_CANT 20
7
8 command_t command_list[COMMAND_MAX_CANT];
9 int commands_added=0;
10
11 command_t * get_command_list() {
12     return command_list;
13 }
14
15 int get_commands_added() {
16     return commands_added;
17 }
18
19 void add_command(char * name, main function, char* helpDescription){
20     if(commands_added<COMMAND_MAX_CANT){
21         command_list[commands_added].name=name;
22         command_list[commands_added].start=function;
23         command_list[commands_added].help=helpDescription;
24         commands_added++;
25     }
26 }
27
28 main get_command(char * name){
29     int i;
30     for(i=0;i<commands_added;i++){
31         if(!strcmp(command_list[i].name, name)){
32             return command_list[i].start;
33         }
34     }
```

```
35     return NULL;
36 }
```

1.38. commands.h

```
1  #ifndef COMMANDS_H
2  #define COMMANDS_H
3
4  typedef int (*main)(int argc, char * argv[]);
5
6
7  struct command_struct {
8      char * name;
9      main start;
10     char * help;
11 };
12
13 typedef struct command_struct command_t;
14
15 void add_command(char * name, main function, char* help);
16 main get_command(char * name);
17
18 char * autocomplete(char * name);
19
20 #endif //COMMANDS_H
```

1.39. shell.c

```
1  #include "shell.h"
2  #include "../std/systemcall.h"
3  #include "../std/stdio.h"
4  #include "../std/string.h"
5
6  #include "commands.h"
7
8  #define NULL 0
9  #define COMAND_LINE_MAX 1000
10 #define EXIT_SYSTEM -15
11
12 #define HISTORY_MAX 20
13
14 #define NAME_MAX_LENGTH 50
15 char name[NAME_MAX_LENGTH] = "unknown";
16 char * pcname = "itba";
17
18 char * strnormalise(char * str) {
19     int j, i;
20     // cambia enters por espacios
21     for (j = 0; str[j] != '\0'; j++) {
22         if (str[j] == '\n' || str[j] == '\t') {
23             str[j] = ' ';
24         }
25     }
26     // elimina espacios del principio
27     while (str[0] == ' ') {
28         str = str + 1;
29     }
30     //elimina espacios del final
31     for (i = strlen(str) - 1; i > 0 && str[i] == ' '; i--) {
32         str[i] = '\0';
33     }
34     //elimina espacios repetidos en el medio
35     for (j = 0; str[j] != '\0'; j++) {
```

```

36         if (str[j] == ' ' && str[j + 1] == ' ') {
37             strcpy(str + j, str + j + 1);
38             j--;
39         }
40     }
41     return str;
42 }
43
44 void printuser() {
45     printf("\x1B[36;1m%@%s:-$ \x1B[0m", name, pcname);
46 }
47
48 int execute(char* comand, int argcant, char * argvec[]) {
49     if (comand[0] == '\0') {
50         return 0;
51     }
52     main_start = get_command(comand);
53     if (start == NULL) {
54         printf("invalid comand: %s\n", comand);
55         return -1;
56     }
57     return start(argcant, argvec);
58 }
59
60 int parseline() {
61     char c;
62     int i = 0;
63     char comand_line[COMAND_LINE_MAX];
64     while ((c = getchar()) != '\n' && i < COMAND_LINE_MAX - 3) {
65         comand_line[i] = c;
66         i++;
67     }
68     if (i >= COMAND_LINE_MAX - 3) {
69         while (getchar() != '\n');
70         printf("\n");
71     }
72     comand_line[i] = '\0';
73     char* comand = strnormalise(comand_line);
74     int argcant = 0;
75     char * argvec[50];
76     int in_quotes = 0;
77     for (i = 0; comand[i] != '\0'; i++) {
78         if (comand[i] == ' ' && !in_quotes) {
79             comand[i] = '\0';
80             argvec[argcant] = &comand[i + 1];
81             argcant++;
82         } else if (comand[i] == '"') {
83             if (!in_quotes) {
84                 argvec[argcant - 1] = &comand[i + 1];
85             }
86             comand[i] = '\0';
87             in_quotes = !in_quotes;
88         }
89     }
90     return execute(comand, argcant, argvec) == EXIT_SYSTEM;
91 }
92
93 int exit_shell(int argc, char* argv[]) {
94     clear_shell();
95     return EXIT_SYSTEM;
96 }
97
98 int echo_shell(int argc, char* argv[]) {
99     int i;
100     for (i = 0; i < argc; i++) {
101         printf("%s\n", argv[i]);
102     }
103     printf("\n");
104     return 0;
105 }
106
107 int getCPUspeed_shell(int argc, char* argv[]) {
108     unsigned long ips;
109     __cpuspeed(&ips);

```

```

110 //printf("Su procesador esta ejecutando %d instrucciones por ←
        segundo.\n", ips);
111 printf("The CPU speed is: %d.%d MHz\n", (ips) / (1024 * 1024), ←
        ((10 * ips) / (1024 * 1024)) % 10);
112 return 0;
113 }
114
115 int clear_shell(int argc, char* argv[]) {
116     printf("\x1B[2J");
117     return 0;
118 }
119
120 int isodd_shell(int argc, char* argv[]) {
121     if (argc < 1) {
122         printf("Usage: isodd <number>\n");
123         return -1;
124     }
125     int number;
126     sscanf(argv[0], "%d", &number);
127
128     if (number % 2 == 0) {
129         printf("The number %d is NOT ODD, its EVEN.", number);
130     } else {
131         printf("The number %d is ODD", number);
132     }
133     printf("\n");
134     return 0;
135 }
136
137 int help_shell(int argc, char* argv[]) {
138     printf("\x1B[33mThese are the commands available: \x1B[0m\n\n");
139     command_t *commands = (command_t *)get_command_list();
140     int i = 0;
141     while (i < get_commands_added()) {
142         printf("\x1B[4m%\x1B[0m\t\t%\n", commands[i].name, commands[←
            i].help);
143         i++;
144     }
145     printf("\nPress CTRL+ALT+SUPR to reboot the system\n");
146     return 0;
147 }
148
149 int rename_shell(int argc, char* argv[]) {
150     if (argc < 1) {
151         printf("Usage: rename <newname>.\n");
152         return -1;
153     }
154     strncpy(name, argv[0], NAME_MAX_LENGTH);
155 }
156
157 static void test_shell_print_usage() {
158     printf("Usage: rename <testcase>.\n");
159     printf("testcases:\n\tprintf\n\tscanf\n");
160 }
161
162 int test_shell(int argc, char* argv[]) {
163     if (argc < 1) {
164         test_shell_print_usage();
165         return -1;
166     }
167     int integer;
168     unsigned int uinteger;
169     char* string;
170     char chr;
171
172
173
174     if (!strcmp(argv[0], "printf")) {
175         printf("\x1B[32mPlease verify the OKval is the same as RETval←
            n\x1B[1mNOTATION: case[OKval]: RETval\x1B[0m");
176         printf("\n");
177         printf("string[hola mundo]: %s", "hola mundo");
178         printf("\n");
179         printf("char[c]: %c", 'c');

```

```

180     printf("\n");
181     printf("integer[-123]: %d", -123);
182     printf("\n");
183     printf("unsigned integer[123]: %u", 123);
184     printf("\n");
185     printf("hexa[FFFFFFFF]: %x", -1);
186     printf("\n");
187 } else if (!strcmp(argv[0], "scanf")) {
188     printf("\x1B[32mPlease verify the OKval is the same as RETval\x1B[1mNOTATION: case [OKval]: RETval\x1B[0m");
189     printf("\n");
190     sscanf("-123", "%d", &integer);
191     printf("integer[-123]: %d", integer);
192     printf("\n");
193     sscanf("123", "%u", &uinteger);
194     printf("unsigned integer[123]: %u", uinteger);
195     printf("\n");
196     sscanf("hello world scanf", "hello %s scanf", string);
197     printf("string[word]: %s", string);
198     printf("\n");
199     sscanf("c", "%c", &chr);
200     printf("char[c]: %c", chr);
201 } else {
202     test_shell_print_usage();
203     return -1;
204 }
205 printf("\n");
206 return 0;
207 }
208
209 void shell_start() {
210     int exit = 0;
211     add_command("test", test_shell, "test cases for functionality");
212     add_command("rename", rename_shell, "changes the name of the user ←
of this pc");
213     add_command("echo", echo_shell, "echoes some text, don't forget ←
the quotes (\") if you use spaces");
214     add_command("clear", clear_shell, "clears the screen");
215     add_command("help", help_shell, "shows help");
216     add_command("isodd", isodd_shell, "tells if the number is odd or ←
not");
217     add_command("exit", exit_shell, "exits the system.");
218     add_command("getCPUspeed", getCPUspeed_shell, "shows actual CPU ←
speed");
219     do {
220         printf("\x1B[33mHi! Whats your name? \x1B[0m");
221         char c = '\0';
222         int i = 0;
223         while ((c = getchar()) != '\n' && i < NAME_MAX_LENGTH) {
224             name[i++] = c;
225         }
226         name[i] = '\0';
227         if (i == NAME_MAX_LENGTH) {
228             while (getchar() != '\n');
229         }
230         printf("\x1B[2J\x1B[33mWelcome to arnix (ARg uNIX) %s!\x1B[0m\x1B[0m\nYou may type \x1B[1mhelp\x1B[0m for more information\n\x1B[0m", name);
231         while (!exit) {
232             printuser();
233             exit = parseline();
234         }
235         exit = 0;
236     } while (1);
237 }

```

1.40. shell.h

```
1 #ifndef SHELL_H
2 #define SHELL_H
3
4 void shell_start();
5
6 #endif /* SHELL_H */
```