Codigos fuente

Arnix

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. 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

```
/*****************************************
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);
```

1.3. 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 $
 *
 */
```

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

```
103         *((t*) (void*) ((ap) - __va_argsiz(t))))
104
105 #endif /* Not RC_INVOKED */
106
107 #endif /* not _STDARG_H_ */
```

1.4. 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 ↩
        licensed
8   * to the University of California by American Telephone and Telegraph
9   * Co. or Unix System Laboratories, Inc. and are reproduced herein ↩
        with
10  * the permission of UNIX System Laboratories, Inc.
11  *
12  * Redistribution and use in source and binary forms, with or without
13  * modification, are permitted provided that the following conditions
14  * are met:
15  * 1. Redistributions of source code must retain the above copyright
16  *    notice, this list of conditions and the following disclaimer.
17  * 2. Redistributions in binary form must reproduce the above ↩
        copyright
18  *    notice, this list of conditions and the following disclaimer in ↩
        the
19  *    documentation and/or other materials provided with the ↩
        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

1.5. 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 }
```

1.6. 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
```

1.7. loader.asm

```
1   global _loader       ; making entry point visible to linker
2   global eokl      ; end of kernel land
3   extern kmain         ; _main is defined elsewhere
4
5
6   ; setting up the Multiboot header - see GRUB docs for details
7   MODULEALIGN equ  1<<0                   ; align loaded modules on page↩
        boundaries
8   MEMINFO     equ  1<<1                   ; provide memory map
9   FLAGS       equ  MODULEALIGN | MEMINFO  ; this is the Multiboot 'flag'↩
        field
10  MAGIC       equ    0x1BADB002           ; 'magic number' lets ↩
        bootloader find the header
```

```
11 CHECKSUM     equ -(MAGIC + FLAGS)         ; checksum required
12
13 section .text
14 align 4
15 MultiBootHeader:
16     dd MAGIC
17     dd FLAGS
18     dd CHECKSUM
19
20     ; reserve initial kernel stack space
21     STACKSIZE equ 0x4000         ; that's 16k.
22
23     _loader:
24     mov esp, stack+STACKSIZE; set up the stack
25     push eax          ; pass Multiboot magic number
26     push ebx          ; pass Multiboot info structure
27
28     call  kmain       ; call kernel proper
29     hlt          ; halt machine should kernel return
30
31 eokl    dd STACKSIZE + stack
32     section .bss
33     align 32
34     stack:
35     resb STACKSIZE       ; reserve 16k stack on a quadword boundary
```

driver

1.8. keyboard.c

```
1  #include "../system/isr.h"
2  #include "../system/in_out.h"
3  #include "../system/keyboardlisteners.h"
4
5  #define KEYBOARD 0x60
6  #define BUFFER_SIZE 100
7
8  #define LSHIFT_KEY_PRESED_SCAN_CODE 42
9  #define LSHIFT_KEY_RELESED_SCAN_CODE 170
10 #define RSHIFT_KEY_PRESED_SCAN_CODE 54
11 #define RSHIFT_KEY_RELESED_SCAN_CODE 182
12
13 #define BLOQ_MAYUS_SCAN_CODE 58
14
15 char array[BUFFER_SIZE];
16
17 buffer_t stdin;
18
19 char * actual_scan_code_table;
20
21 char SCAN_CODE_TABLE[60]={'\x1B','@','1','2','3','4','5','6','7','8','↩
       9','0','-','+','\x08','\t','q','w','e','r','t','y','u','i','o','p'↩
       ,'{','}','\n','@','a','s','d','f','g','h','j','k','l','+','@','@',↩
       '@','@','z','x','c','v','b','n','m',',','.','@','@','@','@',' '};
22 char SHIFT_SCAN_CODE_TABLE[60]={'\x1B','@','!','"','#','$',';','&','?'↩
       ,'/','(',')','_','=','\x08','\t','Q','W','E','R','T','Y','U','I',↩
       'O','P','[',']','\n','@','A','S','D','F','G','H','J','K','L','\\',↩
       '"','@','@','@','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      return 0;
34  }
35
36  int bloq_mayusc_unpresed(){
37      bloq_mayusc=1;
38          add_key_listener(-1,BLOQ_MAYUS_SCAN_CODE, bloq_mayusc_presed);
39      return 0;
40  }
41
42  int shift_presed(){
43      shift++;
44      actual_scan_code_table=SHIFT_SCAN_CODE_TABLE;
45      return 0;
46  }
47
48  int shift_relesed(){
49      shift--;
50      if(shift==0){
51          actual_scan_code_table=SCAN_CODE_TABLE;
52      }
53      return 0;
54  }
55
56  void IRQ1_handler(registers_t reg){
57      int tmp;
58      int i=inb(KEYBOARD);
59      if(activate(i)){
60          tmp=(stdin.end+1)%stdin.size;
61          if(tmp!=stdin.start){
62              char c=actual_scan_code_table[i];
63              if(bloq_mayusc){
64                  if(c>='a' && c<='z'){
65                      c=c+'A'-'a';
66                  }else if(c>='A' && c<='Z'){
67                      c=c+'a'-'A';
68                  }
69              }
70              stdin.array[stdin.end]=c;
71              stdin.end=tmp;
72          } else {
73                      //TODO: beep
74          }
75      }
76  }
77
78  static void reset(){
79      outb(0x64,0xFE);
80  }
81
82  static int cnrl_alt_supr_manager(){
83      reset();
84      return 0;
85  }
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_PRESED_SCAN_CODE, shift_presed)↩
                ;
98          add_key_listener(-1,RSHIFT_KEY_PRESED_SCAN_CODE, shift_presed)↩
                ;
99          add_key_listener(-1,LSHIFT_KEY_RELESED_SCAN_CODE, ↩
                shift_relesed);
100         add_key_listener(-1,RSHIFT_KEY_RELESED_SCAN_CODE, ↩
                shift_relesed);
```

```
101         add_key_listener(-1,BLOQ_MAYUS_SCAN_CODE, bloq_mayusc_unpresed↩
                ();
102
103         add_key_listener(3, 83, cnrl_alt_supr_manager);
104 }
```

1.9. keyboard.h

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

1.10. 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 & ↩
                    ansi_colors[u]);
140         } else if (dec == 4) { /* background */
141             screen_settings = (0x0F & screen_settings) | (ansi_colors[↩
                    u] << 4);
142         }
143     }
144 }
145
146 static void do_scape(char c) {
147     switch (screen_state) {
148         case 1:
149             if (c == '[') {
150                 screen_state = 2;
151                 screen_param_count = 1;
152                 int i = 0;
153                 for (; i <= SCREEN_MAX_PARAM_COUNT; i++) {
154                     screen_param[i] = 0;
155                 }
156             } else {
157                 screen_state = 0;
158             }
159             break;
160         case 2:
161             if (c >= '0' && c <= '9') {
162                 screen_param[screen_param_count - 1] = 10 * ↩
                        screen_param[screen_param_count - 1] + (c - '0');
163             } else if (c == ';') {
164                 screen_param_count++;
165             } else {
166                 switch (c) {
167                     case 'm':
168                         do_scape_m();
169                         break;
170                     case 'J':
171                         do_scape_J();
172                         break;
173                 }
174                 screen_state = 0;
175             }
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 }
```

1.11. 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,↩
        column 0).
14  *      Esc[#;#;...m      Cambia el modo de graficos segun los ↩
      siguientes atributos:
```

```
15  *
16  * Text attributes
17  * 0     All attributes off
18  * 1     Bold on
19  * 4     Underscore (on monochrome display adapter only)
20  * 5     Blink on
21  *
22  * Foreground colors     Background colors
23  * 30    Black           40   Black
24  * 31    Red             41   Red
25  * 32    Green           42   Green
26  * 33    Yellow          43   Yellow
27  * 34    Blue            44       Blue
28  * 35    Magenta         45       Magenta
29  * 36    Cyan            46       Cyan
30  * 37    White           47       White
31  *
32  * Ej: Esc[34;47m (azul en fondo blanco)
33  **/
34 void screen_put(char c);
35
36
37 #endif
```

1.12. 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 }
```

1.13. 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

1.14. 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
```

1.15. 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, 0x0);
42     outb(0xA1, 0x0);
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 }
```

1.16. 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();
```

1.17. 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  }
```

1.18. 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
```

1.19. 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  }
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 }
```

1.20. 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 */
```

1.21. 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 ↩
           lo pone negativo
13         regs.int_no*=-1;
14         }
15     if (interrupt_handlers[regs.int_no] != NULL) {
16         isr_t handler = interrupt_handlers[regs.int_no];
17         handler(regs);
18     }
19 }
20
21 void irq_handler(registers_t regs) {
22     if (regs.int_no >= IRQ8) {
23         outb(PORT_PIC2, SIGNAL_EOI);
24     }
25     outb(PORT_PIC1, SIGNAL_EOI);
26     isr_handler(regs);
27 }
```

1.22. 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 ↩
          pushs.
29 } registers_t;
30
31 typedef void (*isr_t)(registers_t);
32 void register_interrupt_handler(uint8_t n, isr_t handler);
33
34 #endif //ISR_H
```

1.23. keyboardlisteners.c

```
1  #ifndef KEYBOARDLISTENER_H
2  #define KEYBOARDLISTENER_H
3
4  #define MAX_SCAN_CODE 300
5
6  #define CTRL_KEY_PRESED_SCAN_CODE 29
7  #define CTRL_KEY_RELESED_SCAN_CODE 157
8
9  #define ALT_KEY_PRESED_SCAN_CODE 56
10 #define ALT_KEY_RELESED_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
```

1.24. keyboardlisteners.h

```
1  #ifndef KEYBOARDLISTENER_H
2  #define KEYBOARDLISTENER_H
3
4  #define MAX_SCAN_CODE 300
```

```
5
6  #define CTRL_KEY_PRESED_SCAN_CODE 29
7  #define CTRL_KEY_RELESED_SCAN_CODE 157
8
9  #define ALT_KEY_PRESED_SCAN_CODE 56
10 #define ALT_KEY_RELESED_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

1.25. 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 ↩
           parameter.
5      lidt [eax]         ; Load the IDT pointer.
6      ret
7
8  %macro ISR_NOERRCODE 1
9    global isr%1
10   isr%1:
11     cli                         ; Disable interrupts firstly.
12     push byte 0                 ; Push a dummy error code.
13     push byte %1                ; Push the interrupt number.
14     jmp isr_common_stub         ; Go to our common handler code.
15 %endmacro
16
17 ; This macro creates a stub for an ISR which passes it's own
18 ; error code.
19 %macro ISR_ERRCODE 1
20   global isr%1
21   isr%1:
22     cli                         ; Disable interrupts.
23     push byte %1                ; Push the interrupt number
24     jmp isr_common_stub
25 %endmacro
26
27 ; This macro creates a stub for an IRQ - the first parameter is
28 ; the IRQ number, the second is the ISR number it is remapped to.
29 %macro IRQ 2
30   global irq%1
31   irq%1:
32     cli
33     push byte 0
34     push byte %2
35     jmp irq_common_stub
36 %endmacro
37
38 ISR_NOERRCODE 0
39 ISR_NOERRCODE 1
40 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
```

1.26. 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

1.27. 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)↩
            -1){
13          return 1;
14      }
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 }
```

1.28. printf.c

```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];
    char c;

    va_list args;

    va_start(args, formatString);

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

            formatString++;

            switch (*formatString) {
                case 'c':
                    c = va_arg(args, int);
                    putchar(c);
                    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++;
```

```
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 }
```

1.29. 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] != ' ') {
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  }
```

1.30. 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
```

1.31. 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  }
```

1.32. 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 */
```

1.33. systemcall.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
```

1.34. systemcall.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

1.35. 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 }
```

1.36. commands.h

```
#ifndef COMMANDS_H
#define COMMANDS_H

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


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

typedef struct command_struct command_t;

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

char * autocomplete(char * name);

#endif //COMMANDS_H
```

1.37. shell.c

```
#include "shell.h"
#include "../std/systemcall.h"
#include "../std/stdio.h"
#include "../std/string.h"

#include "commands.h"

#define NULL 0
#define COMAND_LINE_MAX 1000
#define EXIT_SYSTEM -15

#define HISTORY_MAX 20

#define NAME_MAX_LENGTH 50
char name[NAME_MAX_LENGTH] = "unknown";
char * pcname = "itba";

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;
```

```
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* command = strnormalise(comand_line);
74      int argcant = 0;
75      char * argvec[50];
76      int in_quotes = 0;
77      for (i = 0; command[i] != '\0'; i++) {
78          if (command[i] == ' ' && !in_quotes) {
79              command[i] = '\0';
80              argvec[argcant] = &command[i + 1];
81              argcant++;
82          } else if (command[i] == '"') {
83              if(!in_quotes) {
84                  argvec[argcant-1] = &command[i + 1];
85              }
86              command[i] = '\0';
87              in_quotes = !in_quotes;
88          }
89      }
90      return execute(command, 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 ↩
        segudo.\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%s\x1B[0m\t\t%s\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: test <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     if (!strcmp(argv[0], "printf")) {
173         printf("\x1B[32mPlease verify the OKval is the same as RETval\↩
            n\x1B[1mNOTATION: case[OKval]: RETval\x1B[0m");
174         printf("\n");
175         printf("string[hola mundo]: %s", "hola mundo");
176         printf("\n");
177         printf("char[c]: %c", 'c');
178         printf("\n");
179         printf("integer[-123]: %d", -123);
180         printf("\n");
181         printf("unsigned integer[123]: %u", 123);
182         printf("\n");
183         printf("hexa[FFFFFFFF]: %x", -1);
184         printf("\n");
185     } else if (!strcmp(argv[0],"scanf")) {
```

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

1.38. shell.h

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