

## contador.c

Page 1/2

```

1  #include "decls.h"
2
3  #define COUNTLEN 20
4  #define TICKS (1ULL << 15)
5  #define DELAY(x) (TICKS << (x))
6  #define USTACK_SIZE 4096
7
8  static volatile char *const VGABUF = (volatile void *) 0xb8000;
9
10 static uintptr_t esp;
11 static uint8_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
12 static uint8_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));
13
14
15 static void exit() {
16     uintptr_t tmp = esp;
17     esp = 0;
18     task_swap(&tmp);
19 }
20
21
22 static void yield() {
23     if (esp)
24         task_swap(&esp);
25 }
26
27 static void contador_yield(unsigned lim, uint8_t linea, char color) {
28     char counter[COUNTLEN] = {'0'}; // ASCII digit counter (RTL).
29
30     while (lim-->0) {
31         char *c = &counter[COUNTLEN];
32         volatile char *buf = VGABUF + 160 * linea + 2 * (80 - COUNTLEN);
33
34         unsigned p = 0;
35         unsigned long long i = 0;
36
37         while (i++ < DELAY(6)) // Usar un entero menor si va demasiado lento.
38             ;
39
40         while (counter[p] == '9') {
41             counter[p++] = '0';
42         }
43
44         if (!counter[p]++) {
45             counter[p] = '1';
46         }
47
48         while (c-- > counter) {
49             *buf++ = *c;
50             *buf++ = color;
51         }
52
53         yield();
54     }
55 }
56
57 void contador_run() {
58     // Configurar stack1 y stack2 con los valores apropiados.
59     uintptr_t *a = (uintptr_t*) stack1 + USTACK_SIZE;
60     a -= 3;
61     a[2] = 0x2F;
62     a[1] = 0;
63     a[0] = 100;
64
65
66     uintptr_t *b = (uintptr_t*) stack2 + USTACK_SIZE;

```

## contador.c

Page 2/2

```

67     b -= 3;
68     b[2] = 0x4F;
69     b[1] = 1;
70     b[0] = 90;
71
72     // Llamada a exit al finalizar contador_yield
73     * (--b) = (uintptr_t)exit;
74
75     // Simulo que el primer swap no es el primero
76     * (--b) = (uintptr_t)contador_yield;
77
78     // Seteo los registros calle save a 0
79     * (--b) = 0;
80     * (--b) = 0;
81     * (--b) = 0;
82     * (--b) = 0;
83
84
85     // Actualizar la variable estática âM-^@M-^XespâM-^@M-^Y para que apunte
86     // al del segundo contador.
87     esp = (uintptr_t) b;
88
89     // Lanzar el primer contador con task_exec.
90     task_exec((uintptr_t) contador_yield, (uintptr_t) a);
91 }

```

## handlers.c

Page 1/2

```

1  #include "decls.h"
2
3  #define RELEASE_CODE 0x80
4  #define PROMPT_CURSOR '_'
5  #define MAX_SIZE 81
6  #define SPACE ' '
7  #define LEFT_SHIFT 42
8  #define RIGHT_SHIFT 54
9  #define BACKSPACE '\b'
10 #define SIMPLE_QUOTATION_MARK '\"'
11 #define ENTER '\n'
12 #define ENIE 164
13
14 /**
15  * Handler para el timer (IRQ0). Escribe un carácter cada segundo.
16  */
17 static const uint8_t hz_ratio = 18; // Default IRQ0 freq (18.222 Hz).
18
19 void timer() {
20     static char chars[MAX_SIZE];
21     static unsigned ticks;
22     static int8_t line = 21;
23     static uint8_t idx = 0;
24
25     if (++ticks % hz_ratio == 0) {
26         chars[idx] = '.';
27         chars[++idx] = '\0';
28         vga_write(chars, line, 0x07);
29     }
30
31     if (idx >= sizeof(chars) - 1) {
32         line++;
33         idx = 0;
34     }
35 }
36
37 /**
38  * Mapa de "scancodes" a caracteres ASCII en un teclado QWERTY.
39  */
40 static unsigned char klayout[128] = {
41     //0-9
42     0, 0, '1', '2', '3', '4', '5', '6', '7', '8',
43     //10-19
44     '9', '0', 0, 0, BACKSPACE, 0, 'q', 'w', 'e', 'r',
45     //20-29
46     't', 'y', 'u', 'i', 'o', 'p', '[' , ']', ENTER, 0,
47     //30-40
48     'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ENIE, SIMPLE_QUOTATION_MARK,
49     //41-50
50     0, 0, 0, 'z', 'x', 'c', 'v', 'b', 'n', 'm',
51     //51-60
52     ',', '.', '-', 0, 0, 0, SPACE, 0, 0, 0};
53
54 static const uint8_t KBD_PORT = 0x60;
55
56 static bool is_shift_pressed(uint8_t scancode) {
57
58     bool released = scancode & RELEASE_CODE;
59     scancode &= ~RELEASE_CODE;
60
61     static bool pressed;
62     if (scancode == RIGHT_SHIFT || scancode == LEFT_SHIFT) {
63         pressed = !released;
64     }
65     return pressed;
66 }

```

## handlers.c

Page 2/2

```

67
68 /**
69  * Handler para el teclado (IRQ1).
70  *
71  * Imprime la letra correspondiente por pantalla.
72  */
73 void keyboard() {
74     uint8_t code;
75     static uint8_t actual_index = 0;
76     static unsigned char kbd_entry_line[MAX_SIZE];
77
78     asm volatile("inb %1,%0" : "=a"(code) : "n"(KBD_PORT));
79
80     int8_t offset = is_shift_pressed(code)? -32 : 0;
81
82     if (code >= sizeof(klayout) || !klayout[code])
83         return;
84
85     if (klayout[code] == BACKSPACE) {
86         if (!actual_index)
87             actual_index=1;
88         kbd_entry_line[actual_index] = SPACE;
89         kbd_entry_line[--actual_index] = PROMPT_CURSOR;
90     } else {
91         kbd_entry_line[actual_index] = klayout[code] + offset;
92         kbd_entry_line[++actual_index] = PROMPT_CURSOR;
93     }
94     vga_write((char*)kbd_entry_line, 19, 0x0A);
95
96 }

```

## interrupts.c

Page 1/2

```

1  #include "decls.h"
2  #include "interrupts.h"
3
4  #define IDT_SIZE 256
5  static struct IDTR idtr;
6  static struct Gate idt[IDT_SIZE];
7
8  // Multiboot siempre define "8" como el segmento de código.
9  // (Ver campo CS en 'info registers' de QEMU.)
10 static const uint8_t KSEG_CODE = 8;
11
12 // Identificador de "Interrupt gate de 32 bits" (ver IA32-3A,
13 // tabla 6-2: IDT Gate Descriptors).
14 static const uint8_t STS_IG32 = 0xE;
15
16 #define outb(port, data) \
17     asm("outb %b0,%w1" : : "a"(data), "d"(port));
18
19 static void irq_remap() {
20     outb(0x20, 0x11);
21     outb(0xA0, 0x11);
22     outb(0x21, 0x20);
23     outb(0xA1, 0x28);
24     outb(0x21, 0x04);
25     outb(0xA1, 0x02);
26     outb(0x21, 0x01);
27     outb(0xA1, 0x01);
28     outb(0x21, 0x0);
29     outb(0xA1, 0x0);
30 }
31
32 void idt_install(uint8_t n, void (*handler)(void)) {
33     uintptr_t addr = (uintptr_t) handler;
34
35     idt[n].rpl = 0;
36     idt[n].type = STS_IG32;
37     idt[n].segment = KSEG_CODE;
38
39     idt[n].off_15_0 = addr & 0xFF;
40     idt[n].off_31_16 = addr >> 16;
41
42     idt[n].present = 1;
43 }
44
45 void idt_init() {
46     // (1) Instalar manejadores ("interrupt service routines").
47     idt_install(T_BRKPT, breakpoint);
48
49     // (2) Configurar ubicación de la IDT.
50     idtr.base = (uintptr_t) idt;
51     idtr.limit = 8 * IDT_SIZE - 1;
52
53     // (3) Activar IDT.
54     asm("lidt %0" : : "m"(idtr));
55 }
56
57 void irq_init() {
58     // (1) Redefinir códigos para IRQs.
59     irq_remap();
60
61     // (2) Instalar manejadores.
62     idt_install(T_TIMER, timer_asm);
63     idt_install(T_KEYBOARD, keyboard_asm);
64     idt_install(T_DIVIDE, divzero);
65
66     // (3) Habilitar interrupciones.

```

## interrupts.c

Page 2/2

```

67     asm("sti");
68 }

```

## kern2.c

Page 1/1

```

1  #include "decls.h"
2  #include "multiboot.h"
3  #include "lib/string.h"
4
5  #define USTACK_SIZE 4096
6
7  void kmain(const multiboot_info_t *mbi) {
8      vga_write("kern2 loading.....", 8, 0x70);
9
10     if (mbi->flags) {
11         char buf[256] = "cmdline: ";
12         char *cmdline = (void *) mbi->cmdline;
13         strlcat(buf, cmdline, sizeof(buf));
14         vga_write(buf, 9, 0x07);
15
16         print_mbinfo(mbi);
17     }
18     two_stacks();
19     two_stacks_c();
20     contador_run();
21
22     idt_init();
23     irq_init();
24     asm("int3");
25
26     int8_t linea;
27     uint8_t color;
28     asm("div %4"
29         : "=a"(linea), "=c"(color)
30         : "0"(18), "1"(0xE0), "b"(0), "d"(0));
31
32     vga_write2("Funciona vga_write2?", linea, color);
33
34     asm("hlt");
35 }
36
37 static uint8_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
38 static uint8_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));
39
40 void two_stacks_c() {
41     // Inicializar al *tope* de cada pila.
42     uintptr_t *a = (uintptr_t*) stack1 + USTACK_SIZE;
43     uintptr_t *b = (uintptr_t*) stack2 + USTACK_SIZE;
44
45     // Preparar, en stack1, la llamada:
46
47     *(a--) = 0x57;
48     *(a--) = 15;
49     *(a) = (uintptr_t) "vga_write() from stack1";
50
51     // Preparar, en s2, la llamada:
52
53     b -= 3;
54     b[2] = 0xD0;
55     b[1] = 16;
56     b[0] = (uintptr_t) "vga_write() from stack2";
57
58     task_exec((uintptr_t) vga_write, (uintptr_t) a);
59
60     asm("movl %0, %%esp; call *%1; movl %%ebp, %%esp"
61         :
62         : "r"(b), "r"(vga_write));
63 }

```

## mbinfo.c

Page 1/1

```

1  #include "decls.h"
2  #include "lib/string.h"
3  #include "multiboot.h"
4
5  #define KB_TO_MB_SHIFT 10 // 1KB*2^10->1MB
6
7  void print_mbinfo(const struct multiboot_info *mbi){
8      char mem[256] = "Physical memory: ";
9      char tmp[64] = {0};
10
11     uint32_t total_size = mbi->mem_upper - mbi->mem_lower;
12     if (fmt_int(total_size>>KB_TO_MB_SHIFT, tmp, sizeof tmp)) {
13         strlcat(mem, tmp, sizeof mem);
14         strlcat(mem, "MiB total", sizeof mem);
15     }
16     memset(tmp, 0, sizeof(tmp));
17     if (fmt_int(mbi->mem_lower, tmp, sizeof tmp)) {
18         strlcat(mem, "(", sizeof mem);
19         strlcat(mem, tmp, sizeof mem);
20         strlcat(mem, " KiB base", sizeof mem);
21     }
22
23     memset(tmp, 0, sizeof(tmp));
24     if (fmt_int(mbi->mem_upper, tmp, sizeof tmp)) {
25         strlcat(mem, "(", sizeof mem);
26         strlcat(mem, tmp, sizeof mem);
27         strlcat(mem, " KiB extended", sizeof mem);
28     }
29
30     vga_write(mem, 10, 0x07);
31 }

```

## write.c

Page 1/1

```

1 #include "multiboot.h"
2 #include "decls.h"
3
4 #define VGABUF ((volatile char *) 0xB8000)
5 #define ROWS 25 // numero de filas de la pantalla
6 #define COLUMNS 80 // numero de columnas de la pantalla
7
8 static size_t int_width(uint64_t val) {
9     size_t width = 0;
10    while (val>0){
11        val/=10;
12        width++;
13    }
14    return width;
15 }
16
17 // Escribe en âM-^@M-^XsâM-^@M-^Y el valor de âM-^@M-^XvalâM-^@M-^Y en base 10 s
18 // i su anchura
19 // es menor que âM-^@M-^XbufsizeâM-^@M-^Y. En ese caso devuelve true, caso de
20 // no haber espacio suficiente no hace nada y devuelve false.
21 bool fmt_int(uint64_t val, char *s, size_t bufsize) {
22     size_t l = int_width(val);
23
24     if (l >= bufsize) // Pregunta: ¿por quÃ© no "l > bufsize"?
25         return false; // Respuesta: para agregar el \0
26
27     for (size_t i = l; i > 0; i--) {
28         char ascii_digit = '0'+val %10;
29         s[i-1]= ascii_digit;
30         val/=10;
31     }
32
33     s[l]='\0';
34     return true;
35 }
36
37 void vga_write(const char *s, int8_t linea, uint8_t color) {
38     if (linea < 0) {
39         linea = ROWS + linea;
40     }
41
42     volatile char* buff = VGABUF + linea * COLUMNS * 2;
43     while (*s != '\0') {
44         *buff++ = *s++;
45         *buff++ = color;
46     }
47 }
48
49 void __attribute__((regparm(2)))
50 vga_write_cyan(const char *s, int8_t linea) {
51     vga_write(s, linea, 0xB0);
52 }
53

```

## boot.S

Page 1/1

```

1 #include "multiboot.h"
2
3 #define KSTACK_SIZE 8192
4
5 .align 4
6 multiboot:
7     .long MULTIBOOT_HEADER_MAGIC
8     .long 0
9     .long -(MULTIBOOT_HEADER_MAGIC)
10
11 .globl _start
12 _start:
13     // Paso 1: Configurar el stack antes de llamar a kmain.
14     movl $0, %ebp
15     movl $kstack_top, %esp
16     push %ebp
17
18     // Paso 2: pasar la informaciÃ³n multiboot a kmain. Si el
19     // kernel no arrancÃ³ vÃ¡ a Multiboot, se debe pasar NULL.
20     // Usar una instrucciÃ³n de comparaciÃ³n (TEST o CMP) para
21     // comparar con MULTIBOOT_BOOTLOADER_MAGIC, pero no usar
22     // un salto a continuaciÃ³n, sino una instruccÃ³n CMOVcc
23     // (copia condicional).
24     movl $0, %ecx
25     cmp $MULTIBOOT_BOOTLOADER_MAGIC, %eax
26     cmovl %ebx, %ecx
27     push %ecx
28
29     call kmain
30 halt:
31     hlt
32     jmp halt
33
34 .data
35 .p2align 12
36 kstack:
37     .space KSTACK_SIZE
38 kstack_top:

```

## funcs.S

Page 1/1

```

1 .text
2
3 .globl vga_write2
4 vga_write2:
5     push %ebp
6     movl %esp, %ebp
7
8     push %ecx
9     push %edx
10    push %eax
11    call vga_write
12
13    leave
14    ret

```

## idt\_entry.S

Page 1/2

```

1 #define PIC1 0x20
2 #define ACK_IRQ 0x20
3
4 .globl ack_irq
5 ack_irq:
6     movl $ACK_IRQ, %eax
7     outb %al, $PIC1
8     iret
9
10 .globl breakpoint
11 breakpoint:
12     // (1) Guardar registros.
13     pusha
14     // (2) Preparar argumentos de la llamada.
15     // vga_write2("Hello, breakpoint", 14, 0xB0)
16     movl $0xB0, %ecx
17     movl $14, %edx
18     movl $breakpoint_msg, %eax
19     // (3) Invocar a vga_write2()
20     call vga_write2
21     // (4) Restaurar registros.
22     popa
23     // (5) Finalizar ejecución del manejador.
24     iret
25
26 .globl timer_asm
27 timer_asm:
28     // Guardar registros.
29     pusha
30     call timer
31     // Restaurar registros.
32     popa
33     jmp ack_irq
34
35 .globl keyboard_asm
36 keyboard_asm:
37     pusha
38     call keyboard
39
40     popa
41     jmp ack_irq
42
43 .globl divzero
44 divzero:
45     // (1) Guardar registros.
46     add $1, %ebx
47     push %eax
48     push %ecx
49     push %edx
50
51     // (2) Preparar argumentos de la llamada.
52     //vga_write_cyan("Se divide por ++ebx", 17);
53
54     movl $17, %edx
55     movl $divzero_msg, %eax
56
57     // (3) Invocar a vga_write_cyan()
58     call vga_write_cyan
59
60     // (4) Restaurar registros.
61     pop %edx
62     pop %ecx
63     pop %eax
64
65     // (5) Finalizar ejecución del manejador.
66     iret

```

## idt\_entry.S

Page 2/2

```

67
68 .data
69 breakpoint_msg:
70     .asciz "Hello, breakpoint"
71
72 divzero_msg:
73     .asciz "Se divide por ++ebx"
74

```

## stacks.S

Page 1/1

```

1  #define USTACK_SIZE 4096
2
3  .data
4      .align 4096
5  stack1:
6      .space USTACK_SIZE
7  stack1_top:
8
9      .p2align 12
10 stack2:
11     .space USTACK_SIZE
12 stack2_top:
13
14 msg1:
15     .asciz "vga_write() from stack1"
16 msg2:
17     .asciz "vga_write() from stack2"
18
19 .text
20 .globl two_stacks
21 two_stacks:
22     // PreÃmbulo estÃndar
23     push %ebp
24     movl %esp, %ebp
25     push %ebx
26
27     // Registros para apuntar a stack1 y stack2.
28     mov $stack1_top, %eax
29     mov $stack2_top, %ebx
30
31     // Cargar argumentos a ambos stacks en paralelo. Ayuda:
32     // usar offsets respecto a %eax ($stack1_top), y lo mismo
33     // para el registro usado para stack2_top.
34     movl $0x17, -4(%eax)
35     movl $0x90, -4(%ebx)
36
37     movl $12, -8(%eax)
38     movl $13, -8(%ebx)
39
40     movl $msg1, -12(%eax)
41     movl $msg2, -12(%ebx)
42
43     // Realizar primera llamada con stack1. Ayuda: usar LEA
44     // con el mismo offset que los Ãltimos MOV para calcular
45     // la direcciÃ³n deseada de ESP.
46     leal -12(%eax), %esp
47     call vga_write
48
49     // Restaurar stack original. Ã¿Es %ebp suficiente?
50     movl %ebp, %esp
51
52     // Realizar segunda llamada con stack2.
53     leal -12(%ebx), %esp
54     call vga_write
55
56     // Restaurar registros callee-saved, si se usaron.
57     pop %ebx
58
59     leave
60     ret

```

tasks.S

Page 1/1

```
1 .data
2
3 .text
4 .globl task_exec
5 task_exec:
6
7     push %ebp
8     movl %esp, %ebp
9
10    movl 8(%ebp), %eax
11    movl 12(%ebp), %esp
12    call *%eax
13
14    leave
15    ret
16
17 .globl task_swap
18 task_swap:
19     push %ebp
20     push %ebx
21     push %edi
22     push %esi
23
24     movl 20(%esp), %eax
25
26     movl %esp, %ecx
27     movl (%eax), %esp
28     movl %ecx, (%eax)
29
30     pop %esi
31     pop %edi
32     pop %ebx
33     pop %ebp
34
35     ret
```

Table of Content

Page 1/1

1	<b>Table of Contents</b>				
2	1 <i>contador.c</i> .....	sheets	1 to	1 ( 1) pages	1- 2 92 lines
3	2 <i>handlers.c</i> .....	sheets	2 to	2 ( 1) pages	3- 4 97 lines
4	3 <i>interrupts.c</i> .....	sheets	3 to	3 ( 1) pages	5- 6 69 lines
5	4 <i>kern2.c</i> .....	sheets	4 to	4 ( 1) pages	7- 7 64 lines
6	5 <i>mbinfo.c</i> .....	sheets	4 to	4 ( 1) pages	8- 8 32 lines
7	6 <i>write.c</i> .....	sheets	5 to	5 ( 1) pages	9- 9 54 lines
8	7 <i>boot.S</i> .....	sheets	5 to	5 ( 1) pages	10- 10 39 lines
9	8 <i>funcs.S</i> .....	sheets	6 to	6 ( 1) pages	11- 11 15 lines
10	9 <i>idt_entry.S</i> .....	sheets	6 to	7 ( 2) pages	12- 13 75 lines
11	10 <i>stacks.S</i> .....	sheets	7 to	7 ( 1) pages	14- 14 61 lines
12	11 <i>tasks.S</i> .....	sheets	8 to	8 ( 1) pages	15- 15 36 lines