

Jun 21, 18 18:09

contador.c

Page 1/2

```

1  #include "decls.h"
2  #include "sched.h"
3
4  #define COUNTLEN 20
5  #define TICKS (1ULL << 15)
6  #define DELAY(x) (TICKS << (x))
7  #define USTACK_SIZE 1024
8
9  static volatile char *const VGABUF = (volatile void *) 0xb8000;
10
11 static uintptr_t esp;
12 static uint32_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
13 static uint32_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));
14
15 static void yield() {
16     if (esp)
17         task_swap(&esp);
18 }
19
20 static void exit() {
21     uintptr_t tmp = esp;
22     esp = 0;
23     task_swap(&tmp);
24 }
25
26 static void contador(unsigned lim, uint8_t linea, char color, bool do_yield){
27     char counter[COUNTLEN] = {'0'}; // ASCII digit counter (RTL).
28
29     while (lim-->0) {
30         char *c = &counter[COUNTLEN];
31         volatile char *buf = VGABUF + 160 * linea + 2 * (80 - COUNTLEN);
32
33         unsigned p = 0;
34         unsigned long long i = 0;
35
36         while (i++ < DELAY(6)) // Usar un entero menor si va demasiado lento.
37             ;
38
39         while (counter[p] == '9') {
40             counter[p++] = '0';
41         }
42
43         if (!counter[p]++) {
44             counter[p] = '1';
45         }
46
47         while (c-- > counter) {
48             *buf++ = *c;
49             *buf++ = color;
50         }
51
52         if (do_yield){
53             yield();
54         }
55     }
56 }
57
58 void contador_round_robin(unsigned lim, uint8_t linea, char color) {
59     contador (lim, linea, color, false);
60     kill_task();
61 }
62
63 static void contador_yield(unsigned lim, uint8_t linea, char color) {
64     contador (lim, linea, color, true);
65 }
66
67 void contador_run() {
68     uintptr_t *a = stack1 + sizeof(stack1);

```

Jun 21, 18 18:09

contador.c

Page 2/2

```
71     uintptr_t *b = stack2 + sizeof(stack2);
72
73     * (--a) = 0x2F;    //Color
74     * (--a) = 0;      //Linea
75     * (--a) = 100;    //Limite
76
77     * (--b) = 0x4F;    //Color
78     * (--b) = 1;      //Linea
79     * (--b) = 90;     //Limite
80
81     //Direccion de retorno de la funcion contador_yield
82     * (--b) = (uintptr_t)exit;
83
84     //Direccion de retorno de la funcion task_swap en la primer iteracion
85     * (--b) = (uintptr_t)contador_yield;
86
87     //Registros calle-saved (ebp, ebx, esi, edi)
88     * (--b) = 0;
89     * (--b) = 0;
90     * (--b) = 0;
91     * (--b) = 0;
92
93     esp = (uintptr_t)b;
94
95     task_exec((uintptr_t) contador_yield, (uintptr_t) a);
96 }
```

Jun 21, 18 17:26

handlers.c

Page 1/3

```

1  #include "decls.h"
2  #include <stdbool.h>
3
4  /**
5   * Handler para el timer (IRQ0). Escribe un caracter cada segundo.
6   */
7  static const uint8_t hz_ratio = 18; // Default IRQ0 freq (18.222 Hz).
8
9  void timer() {
10     static char chars[81];
11     static unsigned ticks;
12     static int8_t line = 21;
13     static uint8_t idx = 0;
14
15     if (++ticks % hz_ratio == 0) {
16         chars[idx] = '.';
17         chars[++idx] = '\0';
18         vga_write(chars, line, 0x07);
19     }
20
21     if (idx >= sizeof(chars) - 1) {
22         line++;
23         idx = 0;
24     }
25 }
26
27 /**
28  * Mapa de "scancodes" a caracteres ASCII en un teclado QWERTY.
29  */
30
31 #define CURSOR '^'
32 #define LEFT_ARROW '=' //Ascii que no se usa
33 #define RIGHT_ARROW '#' //Ascii que no se usa
34 #define CAPSLOCK '!' //Ascii que no se usa
35 #define MAX_WRITE 79
36 #define SPACE ' '
37 #define BACKSPACE '\b'
38 #define ENTER '\n'
39
40
41 static char klayout[128] = {
42     0, 0, '1', '2', '3', '4', '5', '6', '7', '8',
43     '9', '0', 0, 0, BACKSPACE, 0, 'q', 'w', 'e', 'r',
44     't', 'y', 'u', 'i', 'o', 'p', '[', ']', ENTER, 0,
45     'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\',
46     0, 0, 0, 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.',
47     '/', 0, 0, 0, SPACE, CAPSLOCK, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
48     0, 0, 0, 0, 0, 0, 0, LEFT_ARROW, 0, RIGHT_ARROW};
49
50 static const uint8_t KBD_PORT = 0x60;
51
52 static bool use_upper(uint8_t code, int caps_lock) {
53     static bool shift_pressed;
54
55     bool released = code & 0x80;
56     code = code & ~0x80;
57
58     if (code == 42 || code == 54) {
59         shift_pressed = !released;
60     }
61
62     if (caps_lock) {
63         return !shift_pressed;
64     }
65
66     return shift_pressed;
67 }
68
69 /**
70  * Handler para el teclado (IRQ1).

```

Jun 21, 18 17:26

handlers.c

Page 2/3

```

71  *
72  * Imprime la letra correspondiente por pantalla.
73  */
74  void keyboard() {
75      uint8_t code;
76      static char chars[MAX_WRITE];
77      static char chars_selected[MAX_WRITE];
78      static uint8_t idx = 0;
79      static int init = 0;
80      static int caps_lock = 0;
81
82      if (init == 0) {
83          for (int i = 0; i < MAX_WRITE; i++) {
84              chars[i] = SPACE;
85              chars_selected[i] = SPACE;
86          }
87          init = 1;
88      }
89      asm volatile("inb %1,%0" : "=a"(code) : "n"(KBD_PORT));
90
91      int8_t upper_shift = use_upper(code, caps_lock) ? -32 : 0;
92
93      if (code >= sizeof(klayout) || !klayout[code])
94          return;
95
96      if (klayout[code] < 'a' || klayout[code] > 'z') {
97          //No es letra, no aplica mayuscula
98          upper_shift = 0;
99      }
100
101      if (klayout[code] == BACKSPACE) {
102          if (idx == 0) {
103              idx++;
104          }
105          chars_selected[idx] = SPACE;
106          chars[--idx] = SPACE;
107          chars_selected[idx] = CURSOR;
108      } else if (klayout[code] == LEFT_ARROW) {
109
110          chars_selected[idx] = ' ';
111          if (idx == 0) {
112              idx++;
113          }
114          idx--;
115          chars_selected[idx] = CURSOR;
116      } else if (klayout[code] == RIGHT_ARROW) {
117
118          chars_selected[idx] = SPACE;
119          if (idx == MAX_WRITE) {
120              idx--;
121          }
122          idx++;
123          chars_selected[idx] = CURSOR;
124      } else if (klayout[code] == ENTER) {
125
126          //Borra toda la linea
127          for (int i = 0; i < MAX_WRITE; i++) {
128              chars[i] = SPACE;
129              chars_selected[i] = SPACE;
130          }
131          idx = 0;
132          chars_selected[idx] = CURSOR;
133      } else if (klayout[code] == CAPSLOCK) {
134
135          caps_lock = caps_lock ? 0 : 1;
136      } else {
137
138          if (idx >= MAX_WRITE) {
139              chars_selected[idx] = SPACE;
140              while (idx--)

```

Jun 21, 18 17:26

handlers.c

Page 3/3

```
141         chars[idx] = SPACE;
142         idx = 0;
143         chars_selected[idx] = CURSOR;
144     }
145     chars_selected[idx] = SPACE;
146     chars[idx++] = klayout[code] + upper_shift;
147     chars_selected[idx] = CURSOR;
148 }
149
150 vga_write(chars, 19, 0x0A);
151 vga_write(chars_selected, 20, 0x0A);
152 }
```

Jun 21, 18 17:30

interrupts.c

Page 1/1

```

1
2  #include "decls.h"
3  #include "interrupts.h"
4
5  #define IDT_DESCRIPTORs 256
6
7  static struct IDTR idtr;
8  static struct Gate idt[IDT_DESCRIPTORs];
9
10
11  // Multiboot siempre define "8" como el segmento de codigo.
12  // (Ver campo CS en 'info registers' de QEMU.)
13  static const uint8_t KSEG_CODE = 8;
14
15  // Identificador de "Interrupt gate de 32 bits" (ver IA32-3A,
16  // tabla 6-2: IDT Gate Descriptors).
17  static const uint8_t STS_IG32 = 0xE;
18
19
20  #define outb(port, data) \
21      asm("outb %b0,%w1" : : "a"(data), "d"(port));
22
23  static void irq_remap() {
24      outb(0x20, 0x11);
25      outb(0xA0, 0x11);
26      outb(0x21, 0x20);
27      outb(0xA1, 0x28);
28      outb(0x21, 0x04);
29      outb(0xA1, 0x02);
30      outb(0x21, 0x01);
31      outb(0xA1, 0x01);
32      outb(0x21, 0x0);
33      outb(0xA1, 0x0);
34  }
35
36
37  void idt_init(){
38
39      idt_install(T_BRKPT, breakpoint);
40      idt_install(T_DIVIDE, divzero);
41
42      idtr.base = (uintptr_t) idt;
43      idtr.limit = IDT_DESCRIPTORs * 8 - 1;
44
45      asm("lidt %0" : : "m"(idtr)); //activar el uso de la IDT configurada
46  }
47
48
49  void irq_init() {
50      irq_remap();
51
52      idt_install(T_TIMER, timer_asm);
53      idt_install(T_KEYBOARD, keyboard_asm);
54
55      asm("sti");
56  }
57
58
59  void idt_install(uint8_t n, void (*handler)(void)){
60      uintptr_t addr = (uintptr_t) handler;
61
62      idt[n].rpl = 0;
63      idt[n].type = STS_IG32;
64      idt[n].segment = KSEG_CODE;
65
66      idt[n].off_15_0 = addr & 0xFFFF;
67      idt[n].off_31_16 = addr >> 16;
68
69      idt[n].present = 1;
70  }

```

Jun 21, 18 18:09

kern2.c

Page 1/1

```

1  #include "decls.h"
2  #include "sched.h"
3
4  #define MAX 1500
5
6
7  static void contador1() {
8      contador_round_robin(MAX, 2, (char)0xB0);
9  }
10
11 static void contador2() {
12     contador_round_robin(MAX, 3, (char)0xD0);
13 }
14
15 static void contador3() {
16     contador_round_robin(MAX, 4, (char)0xE0);
17 }
18
19 void contador_spawn() {
20     spawn(contador1);
21     spawn(contador2);
22     spawn(contador3);
23 }
24
25 void lab_kernel(){
26
27     contador_run();
28
29     asm("int3");
30
31
32     int8_t linea;
33     uint8_t color;
34
35     asm("div %4"
36         : "=a"(linea), "=c"(color)
37         : "0"(18), "1"(0xE0), "b"(0), "d"(0));
38
39     vga_write2("Funciona vga_write2?", linea, color);
40
41     kill_task();
42 }
43
44 void kmain(const multiboot_info_t *mbi) {
45     vga_write("kern2 loading.....", 8, 0x70);
46
47     if (mbi && mbi->flags) {
48         char buf[256] = "cmdline: ";
49         char *cmdline = (void *) mbi->cmdline;
50
51         strlcat (buf, cmdline, sizeof(buf));
52         vga_write(buf, 9, 0x07);
53
54         print_mbinfo(mbi);
55     }
56
57     two_stacks();
58     two_stacks_c();
59
60     //Ejercicios antes del promocional
61     spawn(lab_kernel);
62
63     //Ejercicio promocional
64     contador_spawn();
65     sched_init();
66
67     idt_init();
68     irq_init();
69 }

```

Jun 15, 18 14:26

mbinfo.c

Page 1/1

```
1  #include "decls.h"
2
3  void print_mbinfo(const struct multiboot_info *mbi){
4      char mem[256] = "Physical memory: ";
5      char tmp[64] = "";
6
7      int size = (mbi->mem_upper - mbi->mem_lower) >> 10;
8      if (fmt_int(size, tmp, sizeof tmp)) {
9          strlcat(mem, tmp, sizeof mem);
10         strlcat(mem, " MiB total", sizeof mem);
11     }
12
13     char tmp2[64] = "";
14     if (fmt_int(mbi->mem_lower, tmp2, sizeof tmp2)) {
15         strlcat(mem, "(", sizeof mem);
16         strlcat(mem, tmp2, sizeof mem);
17         strlcat(mem, " KiB base", sizeof mem);
18     }
19
20     char tmp3[64] = "";
21     if (fmt_int(mbi->mem_upper, tmp3, sizeof tmp3)) {
22         strlcat(mem, ",", sizeof mem);
23         strlcat(mem, tmp3, sizeof mem);
24         strlcat(mem, " KiB extended)", sizeof mem);
25     }
26
27     vga_write(mem, 10, 0x07);
28
29 }
```


Jun 21, 18 18:09

sched.c

Page 1/2

```

1  #include "decls.h"
2  #include "sched.h"
3
4  #define MAX_TASK 10
5  #define IF 0x200
6  #define STACK_SIZE 4096
7
8  static struct Task tasks[MAX_TASK];
9  static struct Task *current;
10 static bool first_iteration = true;
11
12 void sched_init() {
13     for (size_t i = 0; i < MAX_TASK; i++){
14         if (tasks[i].status == READY){
15             current = &tasks[i];
16             current->status = RUNNING;
17             return;
18         }
19     }
20 }
21
22 void spawn(void (*entry)(void)) {
23     struct Task* new_task = NULL;
24     for (size_t i = 0; i < MAX_TASK; i++){
25         if (tasks[i].status == FREE){
26             new_task = &tasks[i];
27             break;
28         }
29     }
30
31     new_task->status = READY;
32
33     uint8_t* stack = &new_task->stack[STACK_SIZE] - sizeof(struct TaskFrame);
34     new_task->frame = (struct TaskFrame *)stack;
35
36     new_task->frame->edi = 0;
37     new_task->frame->esi = 0;
38     new_task->frame->ebp = 0;
39     new_task->frame->esp = 0;
40     new_task->frame->ebx = 0;
41     new_task->frame->edx = 0;
42     new_task->frame->ecx = 0;
43     new_task->frame->eax = 0;
44
45     new_task->frame->cs = 8;
46     new_task->frame->eip = (uint32_t)entry;
47     new_task->frame->eflags = IF;
48 }
49
50 void sched(struct TaskFrame *tf) {
51     struct Task *new = NULL;
52     struct Task *old = current;
53     size_t i;
54
55     for (i = 0; i < MAX_TASK; i++){
56         if (&tasks[i] == old){
57             break;
58         }
59     }
60
61     old->status = READY;
62     while (!new){
63         i++;
64         if (i >= MAX_TASK){
65             i = 0;
66         }
67         if (tasks[i].status == READY){
68             new = &tasks[i];
69             if (first_iteration){
70                 first_iteration = false;

```

Jun 21, 18 18:09

sched.c

Page 2/2

```
71         } else {
72             old->frame = tf;
73         }
74         current = new;
75         current->status = RUNNING;
76
77         asm("movl %0, %%esp\n"
78             "popa\n"
79             "iret\n"
80             :
81             : "g" (current->frame)
82             : "memory");
83     }
84 }
85
86 }
87
88 void kill_task(){
89     current->status = DYING;
90     halt();
91 }
```

Jun 21, 18 17:43

two_stacks_c.c

Page 1/1

```

1  #include "decls.h"
2  #define USTACK_SIZE 1024
3
4  static uint32_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
5  static uint32_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));
6
7  void two_stacks_c() {
8      // Inicializar al *tope* de cada pila.
9      uintptr_t *a = stack1 + sizeof(stack1);
10     uintptr_t *b = stack2 + sizeof(stack2);
11
12     // Preparar, en stack1, la llamada:
13     //vga_write("vga_write() from stack1", 15, 0x57);
14
15     // AYUDA 1: se puede usar alguna forma de pre- o post-
16     // incremento/decremento, segun corresponda:
17     //
18     //      *(a++) = ...
19     //      *(++a) = ...
20     //      *(a--) = ...
21     //      *(--a) = ...
22
23
24     * (--a) = 0x57;
25
26     * (--a) = 15;
27
28     * (--a) = (uintptr_t) "vga_write() from stack1";
29
30     // AYUDA 2: para apuntar a la cadena con el mensaje,
31     // es suficiente con el siguiente cast:
32     //
33     //      ... a ... = (uintptr_t) "vga_write() from stack1";
34
35     // Preparar, en s2, la llamada:
36     //vga_write("vga_write() from stack2", 16, 0xD0);
37
38     // AYUDA 3: para esta segunda llamada, usar esta forma de
39     // asignacion alternativa:
40     b -= 3;
41     b[0] = (uintptr_t) "vga_write() from stack2";
42     b[1] = 16;
43     b[2] = 0xD0;
44
45     // Primera llamada usando task_exec().
46     task_exec((uintptr_t) vga_write, (uintptr_t) a);
47
48     // Segunda llamada con ASM directo. Importante: no
49     // olvidar restaurar el valor de %esp al terminar, y
50     // compilar con: -fasm -fno-omit-frame-pointer.
51
52     asm("movl %0, %%esp; call %1; movl %%ebp, %%esp"
53         : /* no outputs */
54         : "r"(b), "r"(vga_write));
55 }
56

```

Jun 14, 18 17:03

write.c

Page 1/1

```

1  #include "decls.h"
2  #include "multiboot.h"
3
4  #define COLUMNS 80
5  #define ROWS 25
6
7  #define VGABUF ((volatile char *) 0xB8000)
8
9  void vga_write(const char *s, int8_t linea, uint8_t color) {
10     if (linea < 0) {
11         linea = ROWS + linea;
12     }
13     volatile char* buff = VGABUF + linea * COLUMNS * 2;
14     while (*s != '\0') {
15         *buff++ = *s++;
16         *buff++ = color;
17     }
18 }
19
20
21 bool fmt_int(uint64_t val, char *s, size_t bufsize){
22     uint64_t digits = 0;
23     uint64_t aux = val;
24     while (aux > 0){
25         digits++;
26         aux /= 10;
27     }
28
29     if (digits >= bufsize){ //>= para agregar el \0
30         return false;
31     }
32
33     for (int i = digits - 1; i >= 0; i--){
34         s[i] = val % 10 + '0';
35         val /= 10;
36     }
37     s[bufsize - 1] = '\0';
38     return true;
39 }
40
41 void __attribute__((regparm(2)))
42 vga_write_cyan(const char *s, int8_t linea) {
43     vga_write(s, linea, 0xB0);
44 }
45
46

```

Jun 21, 18 17:12

boot.S

Page 1/1

```

1  // boot.S
2
3  #include "multiboot.h"
4
5  #define KSTACK_SIZE 8192
6
7  .align 4
8  multiboot:
9      .long MULTIBOOT_HEADER_MAGIC
10     .long 0
11     .long -(MULTIBOOT_HEADER_MAGIC)
12
13 .globl _start
14 _start:
15     // Paso 1: Configurar el stack antes de llamar a kmain.
16     movl $0, %ebp
17     movl $kstack_end, %esp
18     push %ebp
19
20     // Paso 2: pasar la informacion multiboot a kmain. Si el
21     // kernel no arranco via Multiboot, se debe pasar NULL.
22     movl $0, %ecx
23     CMP $MULTIBOOT_BOOTLOADER_MAGIC, %eax
24     cmove %ebx, %ecx
25     push %ecx
26
27     // Usar una instruccion de comparacion (TEST o CMP) para
28     // comparar con MULTIBOOT_BOOTLOADER_MAGIC, pero no usar
29     // un salto a continuacion, sino una instruccion CMOVcc
30     // (copia condicional).
31     // ...
32
33     call kmain
34
35 .globl halt
36 halt:
37     hlt
38     jmp halt
39
40 .data
41 .p2align 12
42 kstack:
43     .space KSTACK_SIZE
44 kstack_end:
45

```

Jun 15, 18 14:31

funcs.S

Page 1/1

```
1  .globl vga_write2
2  vga_write2:
3      push %ebp
4      movl %esp, %ebp
5
6      push %ecx //color
7      push %edx //linea
8      push %eax //mensaje
9      call vga_write
10
11     leave
12     ret
```

Jun 21, 18 17:31

idt_entry.S

Page 1/2

```

1  #define PIC1 0x20
2  #define ACK_IRQ 0x20
3
4  .globl ack_irq
5  ack_irq:
6      // Indicar que se manejo la interrupcion.
7      movl $ACK_IRQ, %eax
8      outb %al, $PIC1
9      iret
10
11
12  .globl timer_asm
13  timer_asm:
14      // Guardar registros e invocar handler
15      pusha
16      call timer
17
18      // Ack *antes* de llamar a sched()
19      movl $ACK_IRQ, %eax
20      outb %al, $PIC1
21
22      // Llamada a sched con argumento
23      push %esp
24      call sched
25
26      // Retornar (si se volviÃ³ de sched)
27      addl $4, %esp
28      popa
29      iret
30
31  .globl keyboard_asm
32  keyboard_asm:
33      pusha
34      call keyboard
35      popa
36
37      jmp ack_irq
38
39  .globl divzero
40  divzero:
41      // (1) Guardar registros.
42
43      add $1, %ebx
44      push %eax //caller saved
45      push %ecx //caller saved
46      push %edx //caller saved
47
48      // (2) Preparar argumentos de la llamada.
49      //vga_write_cyan("Se divide por ++ebx", 17);
50
51      movl $17, %edx //linea
52      movl $divzero_msg, %eax //mensaje
53
54      // (3) Invocar a vga_write_cyan()
55      call vga_write_cyan
56
57      // (4) Restaurar registros.
58      pop %edx
59      pop %ecx
60      pop %eax
61
62      // (5) Finalizar ejecucion del manejador.
63      iret
64
65
66  .globl breakpoint
67  breakpoint:
68      // (1) Guardar registros.
69      pusha
70

```

Jun 21, 18 17:31

idt_entry.S

Page 2/2

```
71 // (2) Preparar argumentos de la llamada.
72 //vga_write2("Hello, breakpoint", 14, 0xB0);
73
74 movl $0xB0, %ecx //color
75 movl $14, %edx //linea
76 movl $breakpoint_msg, %eax //mensaje
77
78 // (3) Invocar a vga_write2()
79 call vga_write2
80
81 // (4) Restaurar registros.
82 popa
83
84 // (5) Finalizar ejecucion del manejador.
85 iret
86
87 .data
88 breakpoint_msg:
89 .asciz "Hello, breakpoint"
90
91 divzero_msg:
92 .asciz "Se divide por ++ebx"
```


Jun 21, 18 13:31

stacks.S

Page 1/1

```

1  // stacks.S
2  #define USTACK_SIZE 4096
3
4  .data
5      .align 4096
6  stack1:
7      .space USTACK_SIZE
8  stack1_top:
9
10     .p2align 12
11  stack2:
12     .space USTACK_SIZE
13  stack2_top:
14
15  msg1:
16     .asciz "vga_write() from stack1 "
17  msg2:
18     .asciz "vga_write() from stack2 "
19
20
21  .text
22  .align 4
23  .globl two_stacks
24  two_stacks:
25      // Preambulo estandar
26      push %ebp
27      push %ebx
28      movl %esp, %ebp
29
30      // Registros para apuntar a stack1 y stack2.
31      mov $stack1_top, %eax
32      mov $stack2_top, %ebx
33
34      // Cargar argumentos a ambos stacks en paralelo. Ayuda:
35      // usar offsets respecto a %eax ($stack1_top), y lo mismo
36      // para el registro usado para stack2_top.
37      movl $0x17, -4(%eax)
38      movl $0x90, -4(%ebx)
39
40      movl $12, -8(%eax)
41      movl $13, -8(%ebx)
42
43      movl $msg1, -12(%eax)
44      movl $msg2, -12(%ebx)
45
46      // Realizar primera llamada con stack1. Ayuda: usar LEA
47      // con el mismo offset que los ultimos MOV para calcular
48      // la direccion deseada de ESP.
49      leal -12(%eax), %esp
50      call vga_write
51
52      // Restaurar stack original. ¿Es %ebp suficiente?
53      movl %ebp, %esp
54
55      // Realizar segunda llamada con stack2.
56      leal -12(%ebx), %esp
57      call vga_write
58
59      // Restaurar registros callee-saved, si se usaron.
60      movl %ebp, %esp
61      popl %ebx
62      popl %ebp
63
64      ret

```

Jun 21, 18 13:31

tasks.S

Page 1/1

```
1
2 .text
3 .align 4
4 .globl task_exec
5 task_exec:
6     // Preambulo estandar
7     push %ebp
8     movl %esp, %ebp
9
10    movl 8(%ebp), %ecx //entry
11    movl 12(%ebp), %eax //stack
12
13    leal 0(%eax), %esp
14    call *%ecx
15
16    // Restaurar registros callee-saved, si se usaron.
17    movl %ebp, %esp
18    popl %ebp
19
20    ret
21
22
23
24 .globl task_swap
25 task_swap:
26     push %ebp
27     push %ebx
28     push %edi
29     push %esi
30
31     movl 20(%esp), %eax //eax = puntero a esp siguiente
32
33
34     movl %esp, %ecx
35     movl 0(%eax), %esp
36     movl %ecx, 0(%eax)
37
38     pop %esi
39     pop %edi
40     pop %ebx
41     pop %ebp
42
43     ret
```

Jun 21, 18 18:10

Table of Content

Page 1/1

1 Table of Contents

2	1	<i>contador.c</i>	sheets	1 to	2 (2)	pages	1-	2	97	lines
3	2	<i>handlers.c</i>	sheets	3 to	5 (3)	pages	3-	5	153	lines
4	3	<i>interrupts.c</i>	sheets	6 to	6 (1)	pages	6-	6	71	lines
5	4	<i>kern2.c</i>	sheets	7 to	7 (1)	pages	7-	7	70	lines
6	5	<i>mbinfo.c</i>	sheets	8 to	8 (1)	pages	8-	8	30	lines
7	6	<i>sched.c</i>	sheets	9 to	10 (2)	pages	9-	10	92	lines
8	7	<i>two_stacks_c.c</i>	sheets	11 to	11 (1)	pages	11-	11	57	lines
9	8	<i>write.c</i>	sheets	12 to	12 (1)	pages	12-	12	47	lines
10	9	<i>boot.S</i>	sheets	13 to	13 (1)	pages	13-	13	46	lines
11	10	<i>funcs.S</i>	sheets	14 to	14 (1)	pages	14-	14	13	lines
12	11	<i>idt_entry.S</i>	sheets	15 to	16 (2)	pages	15-	16	93	lines
13	12	<i>stacks.S</i>	sheets	17 to	17 (1)	pages	17-	17	65	lines
14	13	<i>tasks.S</i>	sheets	18 to	18 (1)	pages	18-	18	44	lines