decls.h

Page 1/1

```c
#ifndef KERN2_DECL_H
#define KERN2_DECL_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

#define USTACK_SIZE 4096

struct multiboot_info;

// mbinfo.c (ejercicio opcional kern2-meminfo)
void print_mbinfo(const struct multiboot_info *mbi);
bool fmt_int(uint64_t val, char *s, size_t bufsize);

// stacks.S
void two_stacks(void);

// kern2.c
void two_stacks_c(void);

// tasks.S
// Realiza una llamada a "entry" sobre el stack proporcionado.
void task_exec(uintptr_t entry, uintptr_t stack);
void task_swap(uintptr_t *esp);

// contador.c
void contador_run(void);
void round_robin(unsigned lim, uint8_t linea, char color);
void halt();

// interrupts.c
void idt_init(void);
void idt_install(uint8_t code, void (*handler)(void));
void irq_init(void);

// idt_entry.S
void divzero(void);
void breakpoint(void);
void ack_irq(void);
void timer_asm(void);
void keyboard_asm(void);

// handlers.c
void timer(void);
void keyboard(void);

// sched.c
void sched_init(void);

// funcs.S
__attribute__((regparm(3))) void vga_write2(const char *s,
                                            int8_t linea,
                                            uint8_t color);

// write.c
void vga_write(const char *s, int8_t linea, uint8_t color);

__attribute__((regparm(2))) void vga_write_cyan(const char *s, int8_t linea);

#endif
```

interrupts.h

Page 1/1

```c
#ifndef INTERRUPTS_H
#define INTERRUPTS_H

#include <stdint.h>

// IDTR Register (see IA32-3A, Â§6.10 INTERRUPT DESCRIPTOR TABLE).
struct IDTR {
    uint16_t limit;  // Limit
    uint32_t base;   // Base address
} __attribute__((packed));


// Gate descriptors for interrupts (see IA32-3A, Â§6.11 IDT DESCRIPTORS).
struct Gate {
    unsigned off_15_0 : 16;   // Low 16 bits of offset in segment.
    unsigned segment : 16;    // Segment selector (always KSEG_CODE).
    unsigned reserved1 : 8;   // Unused/reserved.
    unsigned type : 4;        // Type (always STS_IG32).
    unsigned system : 1;      // System bit (must be 0).
    unsigned rpl : 2;         // Requestor Privilege Level (always 0).
    unsigned present : 1;     // Present (must be 1 if active).
    unsigned off_31_16 : 16;  // High bits of offset in segment.
};


// x86 exception numbers (see IA32-3A, Â§6.3 SOURCES OF INTERRUPTS).
enum Exception {
    T_DIVIDE = 0,    // Divide error
    T_DEBUG = 1,     // Debug exception
    T_NMI = 2,       // Non-maskable interrupt
    T_BRKPT = 3,     // Breakpoint
    T_OFLOW = 4,     // Overflow
    T_BOUND = 5,     // Bounds check
    T_ILLOP = 6,     // Illegal opcode
    T_DEVICE = 7,    // Device not available
    T_DBLFLT = 8,    // Double fault
    /* T_COPROC */   // Reserved (not generated by recent processors)
    T_TSS = 10,      // Invalid task switch segment
    T_SEGNP = 11,    // Segment not present
    T_STACK = 12,    // Stack exception
    T_GPFLT = 13,    // General protection fault
    T_PGFLT = 14,    // Page fault
    /* T_RES */      // Reserved
    T_FPERR = 16,    // Floating point error
    T_ALIGN = 17,    // Aligment check
    T_MCHK = 18,     // Machine check
    T_SIMDERR = 19,  // SIMD floating point error
};

// kern2 interrupt numbers: we map IRQ0 to 32, and count from there.
enum Interrupt {
    T_TIMER = 32,     // IRQ0
    T_KEYBOARD = 33,  // IRQ1
};

#endif
```

```
/* multiboot.h - Multiboot header file. */
/* Copyright (C) 1999,2003,2007,2008,2009  Free Software Foundation, Inc.
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to
 *  deal in the Software without restriction, including without limitation the
 *  rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 *  sell copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL ANY
 *  DEVELOPER OR DISTRIBUTOR BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 *  WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
 *  IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

/*
 * Changes 2017-07-31 (dato@fi.uba.ar):
 *   - use gcc-defined __ASSEMBLER__ guard, instead of ASM_FILE.
 *   - include <stdint.h> to replace manual typedefs.
 */

#ifndef MULTIBOOT_HEADER
#define MULTIBOOT_HEADER 1

/* How many bytes from the start of the file we search for the header. */
#define MULTIBOOT_SEARCH 8192

/* The magic field should contain this. */
#define MULTIBOOT_HEADER_MAGIC 0x1BADB002

/* This should be in %eax. */
#define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002

/* The bits in the required part of flags field we don’t support. */
#define MULTIBOOT_UNSUPPORTED 0x0000fffc

/* Alignment of multiboot modules. */
#define MULTIBOOT_MOD_ALIGN 0x00001000

/* Alignment of the multiboot info structure. */
#define MULTIBOOT_INFO_ALIGN 0x00000004

/* Flags set in the ’flags’ member of the multiboot header. */

/* Align all boot modules on i386 page (4KB) boundaries. */
#define MULTIBOOT_PAGE_ALIGN 0x00000001

/* Must pass memory information to OS. */
#define MULTIBOOT_MEMORY_INFO 0x00000002

/* Must pass video information to OS. */
#define MULTIBOOT_VIDEO_MODE 0x00000004

/* This flag indicates the use of the address fields in the header. */
#define MULTIBOOT_AOUT_KLUDGE 0x00010000

/* Flags to be set in the ’flags’ member of the multiboot info structure. */

```

```
/* is there basic lower/upper memory information? */
#define MULTIBOOT_INFO_MEMORY 0x00000001
/* is there a boot device set? */
#define MULTIBOOT_INFO_BOOTDEV 0x00000002
/* is the command-line defined? */
#define MULTIBOOT_INFO_CMDLINE 0x00000004
/* are there modules to do something with? */
#define MULTIBOOT_INFO_MODS 0x00000008

/* These next two are mutually exclusive */

/* is there a symbol table loaded? */
#define MULTIBOOT_INFO_AOUT_SYMS 0x00000010
/* is there an ELF section header table? */
#define MULTIBOOT_INFO_ELF_SHDR 0X00000020

/* is there a full memory map? */
#define MULTIBOOT_INFO_MEM_MAP 0x00000040

/* Is there drive info? */
#define MULTIBOOT_INFO_DRIVE_INFO 0x00000080

/* Is there a config table? */
#define MULTIBOOT_INFO_CONFIG_TABLE 0x00000100

/* Is there a boot loader name? */
#define MULTIBOOT_INFO_BOOT_LOADER_NAME 0x00000200

/* Is there a APM table? */
#define MULTIBOOT_INFO_APM_TABLE 0x00000400

/* Is there video information? */
#define MULTIBOOT_INFO_VIDEO_INFO 0x00000800

#ifndef __ASSEMBLER__

#include <stdint.h>

struct multiboot_header {
    /* Must be MULTIBOOT_MAGIC - see above. */
    uint32_t magic;

    /* Feature flags. */
    uint32_t flags;

    /* The above fields plus this one must equal 0 mod 2^32. */
    uint32_t checksum;

    /* These are only valid if MULTIBOOT_AOUT_KLUDGE is set. */
    uint32_t header_addr;
    uint32_t load_addr;
    uint32_t load_end_addr;
    uint32_t bss_end_addr;
    uint32_t entry_addr;

    /* These are only valid if MULTIBOOT_VIDEO_MODE is set. */
    uint32_t mode_type;
    uint32_t width;
    uint32_t height;
    uint32_t depth;
};

/* The symbol table for a.out. */
struct multiboot_aout_symbol_table {
    uint32_t tabsize;
    uint32_t strsize;
```

multiboot.h

Page 3/4

```c
    uint32_t addr;
    uint32_t reserved;
};
typedef struct multiboot_aout_symbol_table multiboot_aout_symbol_table_t;

/* The section header table for ELF. */
struct multiboot_elf_section_header_table {
    uint32_t num;
    uint32_t size;
    uint32_t addr;
    uint32_t shndx;
};
typedef struct multiboot_elf_section_header_table
    multiboot_elf_section_header_table_t;

struct multiboot_info {
    /* Multiboot info version number */
    uint32_t flags;

    /* Available memory from BIOS */
    uint32_t mem_lower;
    uint32_t mem_upper;

    /* "root" partition */
    uint32_t boot_device;

    /* Kernel command line */
    uint32_t cmdline;

    /* Boot-Module list */
    uint32_t mods_count;
    uint32_t mods_addr;

    union {
        multiboot_aout_symbol_table_t aout_sym;
        multiboot_elf_section_header_table_t elf_sec;
    } u;

    /* Memory Mapping buffer */
    uint32_t mmap_length;
    uint32_t mmap_addr;

    /* Drive Info buffer */
    uint32_t drives_length;
    uint32_t drives_addr;

    /* ROM configuration table */
    uint32_t config_table;

    /* Boot Loader Name */
    uint32_t boot_loader_name;

    /* APM table */
    uint32_t apm_table;

    /* Video */
    uint32_t vbe_control_info;
    uint32_t vbe_mode_info;
    uint16_t vbe_mode;
    uint16_t vbe_interface_seg;
    uint16_t vbe_interface_off;
    uint16_t vbe_interface_len;
};
typedef struct multiboot_info multiboot_info_t;

struct multiboot_mmap_entry {
```

multiboot.h

Page 4/4

```c
    uint32_t size;
    uint64_t addr;
    uint64_t len;
#define MULTIBOOT_MEMORY_AVAILABLE 1
#define MULTIBOOT_MEMORY_RESERVED 2
    uint32_t type;
} __attribute__((packed));
typedef struct multiboot_mmap_entry multiboot_memory_map_t;

struct multiboot_mod_list {
    /* the memory used goes from bytes 'mod_start' to 'mod_end-1' inclusive */
    uint32_t mod_start;
    uint32_t mod_end;

    /* Module command line */
    uint32_t cmdline;

    /* padding to take it to 16 bytes (must be zero) */
    uint32_t pad;
};
typedef struct multiboot_mod_list multiboot_module_t;

#endif /* ! __ASSEMBLER__ */

#endif /* ! MULTIBOOT_HEADER */
```

sched.h

Page 1/1

```c
#ifndef KERN2_SCHED_H
#define KERN2_SCHED_H

enum TaskStatus {
    FREE = 0,
    READY,
    RUNNING,
    DYING,
};

struct TaskFrame {
    uint32_t edi;
    uint32_t esi;
    uint32_t ebp;
    uint32_t esp;
    uint32_t ebx;
    uint32_t edx;
    uint32_t ecx;
    uint32_t eax;
    /* below here defined by x86 hardware */
    uint32_t eip;
    uint16_t cs;
    uint16_t padding;
    uint32_t eflags;
} __attribute__((packed));


struct Task {
    uint8_t stack[USTACK_SIZE];
    enum TaskStatus status;
    struct TaskFrame *frame;
};

void sched_init();

void spawn(void (*entry)(void));

void sched(struct TaskFrame *tf);

void kill_current_task();

#endif //KERN2_SCHED_H
```

contador.c

Page 1/2

```c
#include "decls.h"
#include "sched.h"

#define COUNTLEN 20
#define TICKS (1ULL << 15)
#define DELAY(x) (TICKS << (x))

static volatile char *const VGABUF = (volatile void *) 0xb8000;

static uintptr_t esp;
static uint8_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
static uint8_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));


static void exit() {
    uintptr_t tmp = esp;
    esp = 0;
    task_swap(&tmp);
}


static void yield() {
    if (esp)
        task_swap(&esp);
}

static void contador(unsigned lim, uint8_t linea, char color,const bool round_robin_mode) {
    char counter[COUNTLEN] = {’0’};  // ASCII digit counter (RTL).

    while (lim--) {
        char *c = &counter[COUNTLEN];
        volatile char *buf = VGABUF + 160 * linea + 2 * (80 - COUNTLEN);

        unsigned p = 0;
        unsigned long long i = 0;

        while (i++ < DELAY(6))  // Usar un entero menor si va demasiado lento.
            ;

        while (counter[p] == ’9’) {
            counter[p++] = ’0’;
        }

        if (!counter[p]++) {
            counter[p] = ’1’;
        }

        while (c-- > counter) {
            *buf++ = *c;
            *buf++ = color;
        }

        if (!round_robin_mode)
            yield();
    }
    if (round_robin_mode)
        kill_current_task();
}

static void contador_yield(unsigned lim, uint8_t linea, char color) {
    contador(lim, linea, color, false);
}

void round_robin(unsigned lim, uint8_t linea, char color) {
    contador(lim, linea, color, true);
```

contador.c

Page 2/2

```c
}

void contador_run() {
    // Configurar stack1 y stack2 con los valores apropiados.
    uintptr_t *a = (uintptr_t*) stack1 + USTACK_SIZE;
    a -= 3;
    a[2] = 0x2F;
    a[1] = 0;
    a[0] = 200;


    uintptr_t *b = (uintptr_t*) stack2 + USTACK_SIZE;
    b -= 3;
    b[2] = 0x4F;
    b[1] = 1;
    b[0] = 100;

    // Llamada a exit al finalizar contador_yield
    *(--b) = (uintptr_t)exit;

    // Simulo que el primer swap no es el primero
    *(--b) = (uintptr_t)contador_yield;

    // Seteo los registros calle save a 0
    *(--b) = 0;
    *(--b) = 0;
    *(--b) = 0;
    *(--b) = 0;


    // Actualizar la variable estÃ¡tica âM-^@M-^XespâM-^@M-^Y para que apunte
    // al del segundo contador.
    esp = (uintptr_t) b;

    // Lanzar el primer contador con task_exec.
    task_exec((uintptr_t) contador_yield, (uintptr_t) a);
}
```

handlers.c

Page 1/2

```c
#include "decls.h"

#define RELEASE_CODE 0x80
#define PROMPT_CURSOR ’_’
#define MAX_SIZE 81
#define SPACE ’ ’
#define LEFT_SHIFT 42
#define RIGHT_SHIFT 54
#define BACKSPACE ’\b’
#define SIMPLE_QUOTATION_MARK ’\’’
#define ENTER ’\n’
#define ENIE 164

/**
 * Handler para el timer (IRQ0). Escribe un carÃ¡cter cada segundo.
 */
static const uint8_t hz_ratio = 18;  // Default IRQ0 freq (18.222 Hz).

void timer() {
    static char chars[MAX_SIZE];
    static unsigned ticks;
    static int8_t line = 21;
    static uint8_t idx = 0;

    if (++ticks % hz_ratio == 0) {
        chars[idx] = ’.’;
        chars[++idx] = ’\0’;
        vga_write(chars, line, 0x07);
    }

    if (idx >= sizeof(chars) - 1) {
        line++;
        idx = 0;
    }
}

/**
 * Mapa de "scancodes" a caracteres ASCII en un teclado QWERTY.
 */
static unsigned char klayout[128] = {
    //0-9
    0,   0,   ’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’,
    //10-19
    ’9’, ’0’, 0,   0, BACKSPACE, 0, ’q’, ’w’, ’e’, ’r’,
    //20-29
    ’t’, ’y’, ’u’, ’i’, ’o’, ’p’, ’[’, ’]’,   ENTER,   0,
    //30-40
    ’a’, ’s’, ’d’, ’f’, ’g’, ’h’, ’j’, ’k’, ’l’, ENIE, SIMPLE_QUOTATION_MARK,
    //41-50
    0, 0,   0,   ’z’, ’x’, ’c’, ’v’, ’b’, ’n’, ’m’,
    //51-60
    ’,’, ’.’, ’_’, 0, 0, 0, SPACE, 0,0,0};

static const uint8_t KBD_PORT = 0x60;

static bool is_shift_pressed(uint8_t scancode) {

    bool released = scancode & RELEASE_CODE;
    scancode &= ~RELEASE_CODE;

    static bool pressed;
    if (scancode == RIGHT_SHIFT || scancode == LEFT_SHIFT) {
        pressed = !released;
    }
    return pressed;
}
```

handlers.c

```c

/**
 * Handler para el teclado (IRQ1).
 *
 * Imprime la letra correspondiente por pantalla.
 */
void keyboard() {
    uint8_t code;
    static uint8_t actual_index = 0;
    static unsigned char kbd_entry_line[MAX_SIZE];

    asm volatile("inb %1,%0" : "=a"(code) : "n"(KBD_PORT));

    int8_t offset = is_shift_pressed(code)? -32 : 0;

    if (code >= sizeof(klayout) || !klayout[code])
        return;

    if (klayout[code] == BACKSPACE) {
        if (!actual_index)
            actual_index=1;
        kbd_entry_line[actual_index] = SPACE;
        kbd_entry_line[--actual_index] = PROMPT_CURSOR;
    } else {
        kbd_entry_line[actual_index] = klayout[code] + offset;
        kbd_entry_line[++actual_index] = PROMPT_CURSOR;
    }
    vga_write((char*)kbd_entry_line, 19, 0x0A);

}
```

interrupts.c

```c
#include "decls.h"
#include "interrupts.h"

#define IDT_SIZE 256
static struct IDTR idtr;
static struct Gate idt[IDT_SIZE];

// Multiboot siempre define "8" como el segmento de cÃ³digo.
// (Ver campo CS en ‘info registers‘ de QEMU.)
static const uint8_t KSEG_CODE = 8;

// Identificador de "Interrupt gate de 32 bits" (ver IA32-3A,
// tabla 6-2: IDT Gate Descriptors).
static const uint8_t STS_IG32 = 0xE;

#define outb(port, data) \
        asm("outb %b0,%w1" : : "a"(data), "d"(port));

static void irq_remap() {
    outb(0x20, 0x11);
    outb(0xA0, 0x11);
    outb(0x21, 0x20);
    outb(0xA1, 0x28);
    outb(0x21, 0x04);
    outb(0xA1, 0x02);
    outb(0x21, 0x01);
    outb(0xA1, 0x01);
    outb(0x21, 0x0);
    outb(0xA1, 0x0);
}

void idt_install(uint8_t n, void (*handler)(void)) {
    uintptr_t addr = (uintptr_t) handler;

    idt[n].rpl = 0;
    idt[n].type = STS_IG32;
    idt[n].segment = KSEG_CODE;

    idt[n].off_15_0 = addr & 0xFF;
    idt[n].off_31_16 = addr >> 16;

    idt[n].present = 1;
}

void idt_init() {
    // (1) Instalar manejadores ("interrupt service routines").
    idt_install(T_BRKPT, breakpoint);

    // (2) Configurar ubicaciÃ³n de la IDT.
    idtr.base = (uintptr_t) idt;
    idtr.limit = 8 * IDT_SIZE - 1 ;

    // (3) Activar IDT.
    asm("lidt %0" : : "m"(idtr));
}

void irq_init() {
    // (1) Redefinir cÃ³digos para IRQs.
    irq_remap();

    // (2) Instalar manejadores.
    idt_install(T_TIMER, timer_asm);
    idt_install(T_KEYBOARD, keyboard_asm);
    idt_install(T_DIVIDE, divzero);

    // (3) Habilitar interrupciones.
```

interrupts.c

```
    asm("sti");
}
```

kern2.c

```
#include "decls.h"
#include "multiboot.h"
#include "lib/string.h"
#include "sched.h"

static void contador1() {
    round_robin(600, 6, 0x3E);
}

static void contador2() {
    round_robin(200, 7, 0x2A);
}

static void contador3() {
    round_robin(400, 8, 0x1F);
}

void contador_spawn() {
    spawn(contador1);
    spawn(contador2);
    spawn(contador3);
}

static uint8_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
static uint8_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));

void two_stacks_c() {
    // Inicializar al *tope* de cada pila.
    uintptr_t *a = (uintptr_t*) stack1 + USTACK_SIZE;
    uintptr_t *b = (uintptr_t*) stack2 + USTACK_SIZE;

    // Preparar, en stack1, la llamada:

    *(a--) = 0x57;
    *(a--) = 15;
    *(a) = (uintptr_t) "vga_write() from stack1";

    // Preparar, en s2, la llamada:

    b -= 3;
    b[2] = 0xD0;
    b[1] = 16;
    b[0] = (uintptr_t) "vga_write() from stack2";

    task_exec((uintptr_t) vga_write, (uintptr_t) a);

    asm("movl %0, %%esp; call *%1; movl %%ebp, %%esp"
    :
    : "r"(b), "r"(vga_write));
}

void kmain(const multiboot_info_t *mbi) {
    vga_write("kern2 loading.............", 8, 0x70);

    if (mbi->flags) {
        char buf[256] = "cmdline: ";
        char *cmdline = (void *) mbi->cmdline;
        strlcat(buf, cmdline, sizeof(buf));
        vga_write(buf, 9, 0x07);

        print_mbinfo(mbi);
    }

    two_stacks();
    two_stacks_c();

```

kern2.c

```c
    contador_run();

    contador_spawn();
    sched_init();

    idt_init();
    irq_init();
    asm("int3");

    int8_t linea;
    uint8_t color;
    asm("div %4"
        : "=a"(linea), "=c"(color)
        : "0"(18), "1"(0xE0), "b"(0), "d"(0));

    vga_write2("Funciona vga_write2?", linea, color);

    asm("hlt");
}
```

mbinfo.c

```c
#include "decls.h"
#include "lib/string.h"
#include "multiboot.h"

#define KB_TO_MB_SHIFT 10 // 1KB*2^10->1MB

void print_mbinfo(const struct multiboot_info *mbi){
    char mem[256] = "Physical memory: ";
    char tmp[64] = {0};

    uint32_t total_size = mbi->mem_upper - mbi->mem_lower;
    if (fmt_int(total_size>>KB_TO_MB_SHIFT, tmp, sizeof tmp)) {
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, "MiB total", sizeof mem);
    }
    memset(tmp,0, sizeof(tmp));
    if (fmt_int(mbi->mem_lower, tmp, sizeof tmp)) {
        strlcat(mem, " (", sizeof mem);
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, " KiB base", sizeof mem);
    }

    memset(tmp,0, sizeof(tmp));
    if (fmt_int(mbi->mem_upper, tmp, sizeof tmp)) {
        strlcat(mem, ", ", sizeof mem);
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, " KiB extended) ", sizeof mem);
    }

    vga_write(mem, 10, 0x07);
}
```

sched.c

```c
#include "decls.h"
#include "sched.h"

#define MAX_TASK 5
#define IF_FLAG 0x200

static struct Task Tasks[MAX_TASK];
static struct Task *current = NULL;

bool getFreeTask(struct Task **new_task) {
    size_t i=0;
    bool new_free_task=false;
    while ((i<MAX_TASK) && !new_free_task){
        if (Tasks[i].status == FREE){
            (*new_task) = &Tasks[i];
            new_free_task=true;
            }
        i++;
    }
    return new_free_task;
}

void sched_init() {
    size_t  i = 0;
    while (i<MAX_TASK && Tasks[i].status != READY){
        i++;
    }
    if (Tasks[i].status == READY) {
        current = &Tasks[i];
        current->status = RUNNING;
    }
}

void initialize_task(struct Task **task, void (*entry)(void)) {
    (*task)->status = READY;

    uint8_t* stack = &(*task)->stack[USTACK_SIZE] - sizeof(struct TaskFrame);
    (*task)->frame = (struct TaskFrame *)stack;

    (*task)->frame->ebp = 0;
    (*task)->frame->esp = 0;
    (*task)->frame->eax = 0;
    (*task)->frame->ebx = 0;
    (*task)->frame->ecx = 0;
    (*task)->frame->edx = 0;
    (*task)->frame->edi = 0;
    (*task)->frame->esi = 0;
    (*task)->frame->eip = (uint32_t)entry;
    (*task)->frame->cs = 0x8;
    (*task)->frame->eflags = IF_FLAG;
}

void spawn(void (*entry)(void)) {
    struct Task* new_task = NULL;
    bool success = getFreeTask(&new_task);
    if (!success)
        return;
    initialize_task(&new_task, entry);
}


static bool first_call = true;

void sched(struct TaskFrame *tf) {
    bool ready_task_found = false;
    struct Task *previous = current;
```

sched.c

```c

    size_t task_index = 0;
    while ((task_index < MAX_TASK) && (&Tasks[task_index] != previous)){
        task_index++;
    }

    previous->status = READY;

    while (!ready_task_found){
        task_index = (task_index+1) % MAX_TASK;

        if (Tasks[task_index].status == READY){
            ready_task_found = true;

            if (!first_call){
                previous->frame = tf;
            } else
                first_call = false;
            current = &Tasks[task_index];
            current->status = RUNNING;

            asm("movl %0, %%esp\n"
                "popa\n"
                "iret\n"
            :
            : "g"(current->frame)
            : "memory");
        }
    }
}

void kill_current_task(){
    current->status = DYING;
    halt();
}
```

write.c

Page 1/1

```c
#include "multiboot.h"
#include "decls.h"

#define VGABUF ((volatile char *) 0xB8000)
#define ROWS 25 // numero de filas de la pantalla
#define COLUMNS 80 // numero de columnas de la pantalla

static size_t int_width(uint64_t val) {
    size_t width = 0;
    while (val>0){
        val/=10;
        width++;
    }
    return width;
}

// Escribe en âM-^@M-^XsâM-^@M-^Y el valor de âM-^@M-^XvalâM-^@M-^Y en base 10 si su anchura
// es menor que âM-^@M-^XbufsizeâM-^@M-^Y. En ese caso devuelve true, caso de
// no haber espacio suficiente no hace nada y devuelve false.
bool fmt_int(uint64_t val, char *s, size_t bufsize) {
    size_t l = int_width(val);

    if (l >= bufsize)  // Pregunta: Â¿por quÃ© no "l > bufsize"?
                       // Respuesta: para agregar el \0
        return false;

    for (size_t i = l; i > 0; i--) {
        char ascii_digit = ’0’+val %10;
        s[i-1]= ascii_digit;
        val/=10;
    }

    s[l]=’\0’;
    return true;
}

void vga_write(const char *s, int8_t linea, uint8_t color) {
    if (linea < 0) {
        linea = ROWS + linea;
    }

    volatile char* buff = VGABUF + linea * COLUMNS * 2;
    while (*s != ’\0’) {
        *buff++ = *s++;
        *buff++ = color;
    }

}

void __attribute__((regparm(2)))
vga_write_cyan(const char *s, int8_t linea) {
    vga_write(s, linea, 0xB0);
}
```

boot.S

Page 1/1

```asm
#include "multiboot.h"

#define KSTACK_SIZE 8192

.align 4
multiboot:
    .long MULTIBOOT_HEADER_MAGIC
    .long 0
    .long -(MULTIBOOT_HEADER_MAGIC)

.globl _start
_start:
    // Paso 1: Configurar el stack antes de llamar a kmain.
    movl $0, %ebp
    movl $kstack_top, %esp
    push %ebp

    // Paso 2: pasar la informaciÃ³n multiboot a kmain. Si el
    // kernel no arrancÃ³ vÃ-a Multiboot, se debe pasar NULL.
    // Usar una instrucciÃ³n de comparaciÃ³n (TEST o CMP) para
    // comparar con MULTIBOOT_BOOTLOADER_MAGIC, pero no usar
    // un salto a continuaciÃ³n, sino una instrucciÃ³n CMOVcc
    // (copia condicional).
    movl $0, %ecx
    cmp $MULTIBOOT_BOOTLOADER_MAGIC, %eax
    cmove %ebx, %ecx
    push %ecx

    call kmain

.globl halt
halt:
    hlt
    jmp halt

.data
.p2align 12
kstack:
    .space KSTACK_SIZE
kstack_top:
```

funcs.S

```
.text

.globl vga_write2
vga_write2:
    push %ebp
    movl %esp, %ebp

    push %ecx
    push %edx
    push %eax
    call vga_write

    leave
    ret
```

idt_entry.S

```
#define PIC1 0x20
#define ACK_IRQ 0x20

.globl ack_irq
ack_irq:
    movl $ACK_IRQ, %eax
    outb %al, $PIC1
    iret

.globl breakpoint
breakpoint:
    // (1) Guardar registros.
    pusha
    // (2) Preparar argumentos de la llamada.
    // vga_write2("Hello, breakpoint", 14, 0xB0)
    movl $0xB0, %ecx
    movl $14, %edx
    movl $breakpoint_msg, %eax
    // (3) Invocar a vga_write2()
    call vga_write2
    // (4) Restaurar registros.
    popa
    // (5) Finalizar ejecuciÃ³n del manejador.
    iret

.globl timer_asm
timer_asm:
    // Guardar registros.
    pusha
    call timer

    // Ack *antes* de llamar a sched()
    movl $ACK_IRQ, %eax
    outb %al, $PIC1

    // Llamada a sched con argumento
    push %esp
    call sched

    // Retornar (si se volviÃ³ de sched)
    addl $4, %esp
    popa
    iret

.globl keyboard_asm
keyboard_asm:
    pusha
    call keyboard

    popa
    jmp ack_irq

.globl divzero
divzero:
    // (1) Guardar registros.
    add $1, %ebx
    push %eax
    push %ecx
    push %edx

    // (2) Preparar argumentos de la llamada.
    //vga_write_cyan("Se divide por ++ebx", 17);

    movl $17, %edx
    movl $divzero_msg, %eax

```

idt_entry.S

```
    // (3) Invocar a vga_write_cyan()
    call vga_write_cyan

    // (4) Restaurar registros.
    pop %edx
    pop %ecx
    pop %eax

    // (5) Finalizar ejecuciÃ³n del manejador.
    iret

.data
breakpoint_msg:
    .asciz "Hello, breakpoint"

divzero_msg:
    .asciz "Se divide por ++ebx"

```

stacks.S

```
#define USTACK_SIZE 4096

.data
        .align 4096
stack1:
        .space USTACK_SIZE
stack1_top:

        .p2align 12
stack2:
        .space USTACK_SIZE
stack2_top:

msg1:
        .asciz "vga_write() from stack1"
msg2:
        .asciz "vga_write() from stack2"

.text
.globl two_stacks
two_stacks:
        // PreÃ¡mbulo estÃ¡ndar
        push %ebp
        movl %esp, %ebp
        push %ebx

        // Registros para apuntar a stack1 y stack2.
        mov $stack1_top, %eax
        mov $stack2_top, %ebx

        // Cargar argumentos a ambos stacks en paralelo. Ayuda:
        // usar offsets respecto a %eax ($stack1_top), y lo mismo
        // para el registro usado para stack2_top.
        movl $0x17, -4(%eax)
        movl $0x90, -4(%ebx)

        movl $12,  -8(%eax)
        movl $13, -8(%ebx)

        movl $msg1,  -12(%eax)
        movl $msg2, -12(%ebx)

        // Realizar primera llamada con stack1. Ayuda: usar LEA
        // con el mismo offset que los Ãºltimos MOV para calcular
        // la direcciÃ³n deseada de ESP.
        leal -12(%eax), %esp
        call vga_write

        // Restaurar stack original. Â¿Es %ebp suficiente?
        movl %ebp, %esp

        // Realizar segunda llamada con stack2.
        leal -12(%ebx), %esp
        call vga_write

        // Restaurar registros callee-saved, si se usaron.
        pop %ebx

        leave
        ret
```

tasks.S

```
.data

.text
.globl task_exec
task_exec:

    push %ebp
    movl %esp, %ebp

    movl 8(%ebp), %eax
    movl 12(%ebp), %esp
    call *%eax

    leave
    ret

.globl task_swap
task_swap:
    push %ebp
    push %ebx
    push %edi
    push %esi

    movl 20(%esp), %eax

    movl %esp, %ecx
    movl (%eax), %esp
    movl %ecx, (%eax)

    pop %esi
    pop %edi
    pop %ebx
    pop %ebp

    ret
```

Table of Content

Table of Contents