Libs

stdbool.h

```
#ifndef __LIB_STDBOOL_H
#define __LIB_STDBOOL_H

#define bool    _Bool
#define true    1
#define false   0
#define __bool_true_false_are_defined   1

#endif /* lib/stdbool.h */
```

stddef.h

```
#ifndef __LIB_STDDEF_H
#define __LIB_STDDEF_H

#define NULL ((void *) 0)
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *) 0)->MEMBER)

/* GCC predefines the types we need for ptrdiff_t and size_t,
   so that we don't have to guess. */
typedef __PTRDIFF_TYPE__ ptrdiff_t;
typedef __SIZE_TYPE__ size_t;

#endif /* lib/stddef.h */
```

stdint.h

```
#ifndef __LIB_STDINT_H
#define __LIB_STDINT_H

typedef signed char int8_t;
#define INT8_MAX 127
#define INT8_MIN (-INT8_MAX - 1)

typedef signed short int int16_t;
#define INT16_MAX 32767
#define INT16_MIN (-INT16_MAX - 1)
```

```
typedef signed int int32_t;
#define INT32_MAX 2147483647
#define INT32_MIN (-INT32_MAX - 1)

typedef signed long long int int64_t;
#define INT64_MAX 9223372036854775807LL
#define INT64_MIN (-INT64_MAX - 1)

typedef unsigned char uint8_t;
#define UINT8_MAX 255

typedef unsigned short int uint16_t;
#define UINT16_MAX 65535

typedef unsigned int uint32_t;
#define UINT32_MAX 4294967295U

typedef unsigned long long int uint64_t;
#define UINT64_MAX 18446744073709551615ULL

typedef int32_t intptr_t;
#define INTPTR_MIN INT32_MIN
#define INTPTR_MAX INT32_MAX

typedef uint32_t uintptr_t;
#define UINTPTR_MAX UINT32_MAX

typedef int64_t intmax_t;
#define INTMAX_MIN INT64_MIN
#define INTMAX_MAX INT64_MAX

typedef uint64_t uintmax_t;
#define UINTMAX_MAX UINT64_MAX

#define PTRDIFF_MIN INT32_MIN
#define PTRDIFF_MAX INT32_MAX

#define SIZE_MAX UINT32_MAX

#endif /* lib/stdint.h */
```

string.c

```
/*
```

```
 * Part of the Pintos project (http://pintos-os.org/).
 *
 * Copyright (C) 2004, 2005, 2006 Board of Trustees, Leland Stanford
 * Jr. University.  All rights reserved.
 *
 * 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 THE AUTHORS OR COPYRIGHT HOLDERS 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):
 *   - delete #include <debug.h>, #define ASSERT as ((void) 0)
 */

#include "string.h"
#define ASSERT(x) ((void) 0)

/* Copies SIZE bytes from SRC to DST, which must not overlap.
   Returns DST. */
void *memcpy (void *dst_, const void *src_, size_t size) {
    unsigned char *dst = dst_;
    const unsigned char *src = src_;

    ASSERT (dst != NULL || size == 0);
    ASSERT (src != NULL || size == 0);

    while (size-- > 0)
        *dst++ = *src++;

    return dst_;
```

```
}

/* Copies SIZE bytes from SRC to DST, which are allowed to
   overlap.  Returns DST. */
void *memmove (void *dst_, const void *src_, size_t size) {
    unsigned char *dst = dst_;
    const unsigned char *src = src_;

    ASSERT (dst != NULL || size == 0);
    ASSERT (src != NULL || size == 0);

    if (dst < src)
    {
        while (size-- > 0)
            *dst++ = *src++;
    }
    else
    {
        dst += size;
        src += size;
        while (size-- > 0)
            *--dst = *--src;
    }

    return dst;
}

/* Find the first differing byte in the two blocks of SIZE bytes
   at A and B.  Returns a positive value if the byte in A is
   greater, a negative value if the byte in B is greater, or zero
   if blocks A and B are equal. */
int memcmp (const void *a_, const void *b_, size_t size) {
    const unsigned char *a = a_;
    const unsigned char *b = b_;

    ASSERT (a != NULL || size == 0);
    ASSERT (b != NULL || size == 0);

    for (; size-- > 0; a++, b++)
        if (*a != *b)
            return *a > *b ? +1 : -1;
    return 0;
}

/* Finds the first differing characters in strings A and B.
   Returns a positive value if the character in A (as an unsigned
```

```
   char) is greater, a negative value if the character in B (as
   an unsigned char) is greater, or zero if strings A and B are
   equal. */
int strcmp (const char *a_, const char *b_) {
    const unsigned char *a = (const unsigned char *) a_;
    const unsigned char *b = (const unsigned char *) b_;

    ASSERT (a != NULL);
    ASSERT (b != NULL);

    while (*a != '\0' && *a == *b)
    {
        a++;
        b++;
    }

    return *a < *b ? -1 : *a > *b;
}

/* Returns a pointer to the first occurrence of CH in the first
   SIZE bytes starting at BLOCK.  Returns a null pointer if CH
   does not occur in BLOCK. */
void *memchr (const void *block_, int ch_, size_t size) {
    const unsigned char *block = block_;
    unsigned char ch = ch_;

    ASSERT (block != NULL || size == 0);

    for (; size-- > 0; block++)
        if (*block == ch)
            return (void *) block;

    return NULL;
}

/* Finds and returns the first occurrence of C in STRING, or a
   null pointer if C does not appear in STRING.  If C == '\0'
   then returns a pointer to the null terminator at the end of
   STRING. */
char *strchr (const char *string, int c_) {
    char c = c_;

    ASSERT (string != NULL);

    for (;;)
        if (*string == c)
```

```
            return (char *) string;
        else if (*string == '\0')
            return NULL;
        else
            string++;
}

/* Returns the length of the initial substring of STRING that
   consists of characters that are not in STOP. */
size_t strcspn (const char *string, const char *stop) {
    size_t length;

    for (length = 0; string[length] != '\0'; length++)
        if (strchr (stop, string[length]) != NULL)
            break;
    return length;
}

/* Returns a pointer to the first character in STRING that is
   also in STOP.  If no character in STRING is in STOP, returns a
   null pointer. */
char *strpbrk (const char *string, const char *stop) {
    for (; *string != '\0'; string++)
        if (strchr (stop, *string) != NULL)
            return (char *) string;
    return NULL;
}

/* Returns a pointer to the last occurrence of C in STRING.
   Returns a null pointer if C does not occur in STRING. */
char *strrchr (const char *string, int c_) {
    char c = c_;
    const char *p = NULL;

    for (; *string != '\0'; string++)
        if (*string == c)
            p = string;
    return (char *) p;
}

/* Returns the length of the initial substring of STRING that
   consists of characters in SKIP. */
size_t strspn (const char *string, const char *skip) {
    size_t length;

    for (length = 0; string[length] != '\0'; length++)
```

```
        if (strchr (skip, string[length]) == NULL)
            break;
    return length;
}

/* Returns a pointer to the first occurrence of NEEDLE within
   HAYSTACK.  Returns a null pointer if NEEDLE does not exist
   within HAYSTACK. */
char *strstr (const char *haystack, const char *needle) {
    size_t haystack_len = strlen (haystack);
    size_t needle_len = strlen (needle);

    if (haystack_len >= needle_len)
    {
        size_t i;

        for (i = 0; i <= haystack_len - needle_len; i++)
            if (!memcmp (haystack + i, needle, needle_len))
                return (char *) haystack + i;
    }

    return NULL;
}

/* Breaks a string into tokens separated by DELIMITERS.  The
   first time this function is called, S should be the string to
   tokenize, and in subsequent calls it must be a null pointer.
   SAVE_PTR is the address of a `char *' variable used to keep
   track of the tokenizer's position.  The return value each time
   is the next token in the string, or a null pointer if no
   tokens remain.

   This function treats multiple adjacent delimiters as a single
   delimiter.  The returned tokens will never be length 0.
   DELIMITERS may change from one call to the next within a
   single string.

   strtok_r() modifies the string S, changing delimiters to null
   bytes.  Thus, S must be a modifiable string.  String literals,
   in particular, are *not* modifiable in C, even though for
   backward compatibility they are not `const'.

   Example usage:

   char s[] = "  String to  tokenize. ";
   char *token, *save_ptr;
```

```
   for (token = strtok_r (s, " ", &save_ptr); token != NULL;
        token = strtok_r (NULL, " ", &save_ptr))
     printf ("'%s'\n", token);

   outputs:

     'String'
     'to'
     'tokenize.'
*/
char *strtok_r (char *s, const char *delimiters, char **save_ptr) {
    char *token;

    ASSERT (delimiters != NULL);
    ASSERT (save_ptr != NULL);

    /* If S is nonnull, start from it.
       If S is null, start from saved position. */
    if (s == NULL)
        s = *save_ptr;
    ASSERT (s != NULL);

    /* Skip any DELIMITERS at our current position. */
    while (strchr (delimiters, *s) != NULL)
    {
        /* strchr() will always return nonnull if we're searching
           for a null byte, because every string contains a null
           byte (at the end). */
        if (*s == '\0')
        {
            *save_ptr = s;
            return NULL;
        }

        s++;
    }

    /* Skip any non-DELIMITERS up to the end of the string. */
    token = s;
    while (strchr (delimiters, *s) == NULL)
        s++;
    if (*s != '\0')
    {
        *s = '\0';
        *save_ptr = s + 1;
```

```
    }
    else
        *save_ptr = s;
    return token;
}

/* Sets the SIZE bytes in DST to VALUE. */
void *memset (void *dst_, int value, size_t size) {
    unsigned char *dst = dst_;

    ASSERT (dst != NULL || size == 0);

    while (size-- > 0)
        *dst++ = value;

    return dst_;
}

/* Returns the length of STRING. */
size_t strlen (const char *string) {
    const char *p;

    ASSERT (string != NULL);

    for (p = string; *p != '\0'; p++)
        continue;
    return p - string;
}

/* If STRING is less than MAXLEN characters in length, returns
   its actual length.  Otherwise, returns MAXLEN. */
size_t strnlen (const char *string, size_t maxlen) {
    size_t length;

    for (length = 0; string[length] != '\0' && length < maxlen; length++)
        continue;
    return length;
}

/* Copies string SRC to DST.  If SRC is longer than SIZE - 1
   characters, only SIZE - 1 characters are copied.  A null
   terminator is always written to DST, unless SIZE is 0.
   Returns the length of SRC, not including the null terminator.

   strlcpy() is not in the standard C library, but it is an
   increasingly popular extension.  See
```

```
   http://www.courtesan.com/todd/papers/strlcpy.html for
   information on strlcpy(). */
size_t strlcpy (char *dst, const char *src, size_t size) {
    size_t src_len;

    ASSERT (dst != NULL);
    ASSERT (src != NULL);

    src_len = strlen (src);
    if (size > 0)
    {
        size_t dst_len = size - 1;
        if (src_len < dst_len)
            dst_len = src_len;
        memcpy (dst, src, dst_len);
        dst[dst_len] = '\0';
    }
    return src_len;
}

/* Concatenates string SRC to DST.  The concatenated string is
   limited to SIZE - 1 characters.  A null terminator is always
   written to DST, unless SIZE is 0.  Returns the length that the
   concatenated string would have assuming that there was
   sufficient space, not including a null terminator.

   strlcat() is not in the standard C library, but it is an
   increasingly popular extension.  See
   http://www.courtesan.com/todd/papers/strlcpy.html for
   information on strlcpy(). */
size_t strlcat (char *dst, const char *src, size_t size) {
    size_t src_len, dst_len;

    ASSERT (dst != NULL);
    ASSERT (src != NULL);

    src_len = strlen (src);
    dst_len = strlen (dst);
    if (size > 0 && dst_len < size)
    {
        size_t copy_cnt = size - dst_len - 1;
        if (src_len < copy_cnt)
            copy_cnt = src_len;
        memcpy (dst + dst_len, src, copy_cnt);
        dst[dst_len + copy_cnt] = '\0';
    }
```

```
    return src_len + dst_len;
}
```

string.h

```
/*
 * Part of the Pintos project (http://pintos-os.org/).
 *
 * Copyright (C) 2004, 2005, 2006 Board of Trustees, Leland Stanford
 * Jr. University.  All rights reserved.
 *
 * 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 THE AUTHORS OR COPYRIGHT HOLDERS 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):
 *   - rename include guard not to start with an underscore, to appease clang
 * Changes 2017-09-20 (dato@fi.uba.ar):
 *   - remove non-functional macros for str[n]cpy, str[n]cat, strtok.
 */

#ifndef STRING_H
#define STRING_H

#include "stddef.h"

/* Standard. */
```

```
void *memcpy(void *, const void *, size_t);
void *memmove(void *, const void *, size_t);
int memcmp(const void *, const void *, size_t);
int strcmp(const char *, const char *);
void *memchr(const void *, int, size_t);
char *strchr(const char *, int);
size_t strcspn(const char *, const char *);
char *strpbrk(const char *, const char *);
char *strrchr(const char *, int);
size_t strspn(const char *, const char *);
char *strstr(const char *, const char *);
void *memset(void *, int, size_t);
size_t strlen(const char *);

/* Extensions. */
size_t strlcpy(char *, const char *, size_t);
size_t strlcat(char *, const char *, size_t);
char *strtok_r(char *, const char *, char **);
size_t strnlen(const char *, size_t);

#endif /* lib/string.h */
```

Src

boot.S

```
#include "multiboot.h"

#define KSTACK_SIZE 8192
#define FLAGS 0
#define CRC ( -(MULTIBOOT_HEADER_MAGIC + FLAGS) )

.align 4
multiboot:
    .long MULTIBOOT_HEADER_MAGIC
    .long FLAGS
    .long CRC

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

```
    movl %esp, %ebp

    // Paso 2: pasar la información multiboot a kmain. Si el
    // kernel no arrancó vía Multiboot, se debe pasar NULL.
    //
    // Usar una instrucción de comparación (TEST o CMP) para
    // comparar con MULTIBOOT_BOOTLOADER_MAGIC, pero no usar
    // un salto a continuación, sino una instrucción CMOVcc
    // (copia condicional).
    mov $0, %ecx
    // si arranco con multiboot -> ZF = 1
    test MULTIBOOT_BOOTLOADER_MAGIC, %eax
    cmovne %ecx, %ebx
    push %ebx
    call kmain
halt:
    hlt
    jmp halt

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

contador.c

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

#define COUNTLEN 20
#define TICKS (1ULL << 15)
#define DELAY(x) (TICKS << (x))
#define USTACK_SIZE 4096

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)));
//------------------------------------------------------------------------
// EXIT
//------------------------------------------------------------------------
```

```
static void exit() {
    uintptr_t tmp = esp;
    esp = 0;
    if (tmp)
        task_swap(&tmp);
}
//-----------------------------------------------------------------------------
// YIELD
//-----------------------------------------------------------------------------
static void yield() {
    if (esp)
        task_swap(&esp);
}
//-----------------------------------------------------------------------------
// CONTADOR YIELD
//-----------------------------------------------------------------------------
static void contador_yield(unsigned lim, uint8_t linea, char color) {
    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;
        }
        yield();
    }
}
//-----------------------------------------------------------------------------
// CONTADOR 1
```

```
//--------------------------------------------------------------------------
static void contador1() {
    contador_yield(50000000, 2, 0x2F);
}
//--------------------------------------------------------------------------
// CONTADOR 2
//--------------------------------------------------------------------------
static void contador2() {
    contador_yield(50000000, 3, 0x6F);
}
//--------------------------------------------------------------------------
// CONTADOR 3
//--------------------------------------------------------------------------
static void contador3() {
    contador_yield(50000000, 4, 0x4F);
}
//--------------------------------------------------------------------------
// CONTADOR SPAWN
//--------------------------------------------------------------------------
void contador_spawn() {
    spawn(contador1);
    spawn(contador2);
    spawn(contador3);
}
//--------------------------------------------------------------------------
// CONTADOR RUN
//--------------------------------------------------------------------------
void contador_run() {
    // Configurar stack1 y stack2 con los valores apropiados.
    uintptr_t *a = (uintptr_t*) &stack1[USTACK_SIZE];
    uintptr_t *b = (uintptr_t*) &stack2[USTACK_SIZE];

    //contador_yield(100, 0, 0x2F);
    *(--a) = 0x2F;  // color
    *(--a) = 0;  // linea
    *(--a) = 100;  // numero

    /*
     la configuración del segundo contador es más compleja, y seguramente sea
     mejor realizarla tras implementar task_swap(); pues se debe crear
     artificialmente el stack tal y como lo hubiera preparado esta función.
    */
    //contador_yield(100, 1, 0x4F);
    *(--b) = 0x4F;  // color
    *(--b) = 1;     // linea
    *(--b) = 10;   // numero
```

```
    *(--b) = (uintptr_t) exit;                //ret falso
    *(--b) = (uintptr_t) contador_yield;      //ret cominezo
    *(--b) = 0;     //push %edi
    *(--b) = 0;     //push %ebp
    *(--b) = 0;     //push %esi
    *(--b) = 0;     //push %ebx
    // Actualizar la variable estática 'esp' 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);
}
//------------------------------------------------------------------------------
```

decls.h

```
#ifndef KERN2_DECL_H
#define KERN2_DECL_H

#include "lib/stdint.h"
#include "lib/stdbool.h"
#include "lib/stddef.h"

struct multiboot_info;

// mbinfo.c (ejercicio opcional kern2-meminfo)
void print_mbinfo(const struct multiboot_info *mbi);

// stacks.S
void two_stacks(void);

// kern2.c
void two_stacks_c(void);

// tasks.S
void task_exec(uintptr_t entry, uintptr_t stack);
void task_swap(uintptr_t *esp);

// contador.c
void contador_run(void);
void contador_spawn();
```

```
// 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);

// 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);
bool fmt_int(uint64_t val, char *s, size_t bufsize);
void print(uint64_t value, int8_t line);
__attribute__((regparm(2))) void vga_write_cyan(const char *s, int8_t linea);

#endif
```

funcs.S

```
.globl vga_write2
vga_write2:
        push %ebp
        movl %esp, %ebp
        /*
        On x86-32 targets, the regparm attribute causes the compiler to pass
        arguments number one to number if they are of integral type in registers
        EAX, EDX, and ECX instead of on the stack. Functions that take a
        variable number of arguments continue to be passed all of their
        arguments on the stack.
        Beware that on some ELF systems this attribute is unsuitable for global
        functions in shared libraries with lazy binding (which is the default).
        Lazy binding sends the first call via resolving code in the loader,
```

```
        which might assume EAX, EDX and ECX can be clobbered, as per the
        standard calling conventions. Solaris 8 is affected by this. Systems
        with the GNU C Library version 2.1 or higher and FreeBSD are believed to
        be safe since the loaders there save EAX, EDX and ECX. (Lazy binding can
        be disabled with the linker or the loader if desired, to avoid the
        problem.)
        */
        push %ecx         // 3rd argument (color)
        push %edx         // 2nd argument (linea)
        push %eax         // 1st argument (msg)
        call vga_write
        // movl %esp, %ebp
        leave
        ret
```

handlers.c

```
#include "decls.h"

static unsigned ticks;
//-------------------------------------------------------------------------------
// TIMER
//-------------------------------------------------------------------------------
void timer() {
    if (++ticks == 15) {
        vga_write("Transcurrieron 15 ticks", 20, 0x07);
    }
}
//-------------------------------------------------------------------------------

/**
 * Mapa de "scancodes" a caracteres ASCII en un teclado QWERTY.
 */
static char klayout[128] = {
    0,   0,   '1', '2', '3', '4', '5', '6', '7', '8',            // 0-9
    '9', '0', 0,   0,   0,   0,   'q', 'w', 'e', 'r',            // 10-19
    't', 'y', 'u', 'i', 'o', 'p', 0,   0,   0,   0,              // 20-29
    'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 0,   0,         // 30-40
    0,   0,   0,   'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.'};  // 41-52

static const uint8_t KBD_PORT = 0x60;

static bool use_upper(uint8_t code) {
    // return false;
```

```
    static bool shift_pressed;

    bool released = code & 0x80;
    code = code & ~0x80;

    if (code == 42 || code == 54) {
        shift_pressed = !released;
    }

    return shift_pressed;
}

/**
 * Handler para el teclado (IRQ1).
 *
 * Imprime la letra correspondiente por pantalla.
 */
void keyboard() {
    uint8_t code;
    static char chars[81];
    static uint8_t idx = 0;

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

    int8_t upper_shift = use_upper(code) ? -32 : 0;

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

    if (idx == 80) {
        while (idx--)
            chars[idx] = ' ';
    }

    chars[idx++] = klayout[code] + upper_shift;
    vga_write(chars, 19, 0x0A);
}
```

idt_entry.S

```
//-----------------------------------------------------------------------------
// BREAKPOINT
//-----------------------------------------------------------------------------
```

```
.globl breakpoint
breakpoint:
        // (1) Guardar registros.
        push %eax
        push %edx
        push %ecx
        // (2) Preparar argumentos de la llamada.
        // vga_write2("Hello, breakpoint", 14, 0xB0);
        movl $0xB0, %ecx                    // 3rd argument (color)
        movl $14, %edx                      // 2nd argument (linea)
        movl $breakpoint_msg, %eax          // 1st argument (msg)
        // (3) Invocar a vga_write2()
        call vga_write2
        // (4) Restaurar registros.
        pop %ecx
        pop %edx
        pop %eax
        // (5) Finalizar ejecución del manejador.
        iret

.data
breakpoint_msg:
        .asciz "Hello, breakpoint"
//-----------------------------------------------------------------------------
// ACK IRQ
//-----------------------------------------------------------------------------
#define PIC1 0x20
#define ACK_IRQ 0x20
.globl ack_irq
ack_irq:
        // Indicar que se manejó la interrupción.
        movl $ACK_IRQ, %eax
        outb %al, $PIC1
        call keyboard
        iret
//-----------------------------------------------------------------------------
// TIMER ASM
//-----------------------------------------------------------------------------
.globl timer_asm
timer_asm:
        // Guardar registros e invocar handler
        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 volvió de sched)
        addl $4, %esp
        popa
        iret
//-----------------------------------------------------------------------------
// DIVZERO
//-----------------------------------------------------------------------------
.globl divzero
divzero:
        // El manejador debe, primero, incrementar el valor de %ebx, de manera
        // que cuando se reintente la instrucción, ésta tenga éxito.
        mov $1, %ebx
        // (1) Guardar registros.
        push %eax
        push %edx
        push %ecx
        // vga_write_cyan("Se divide por ++ebx", 17);
        movl $17, %edx    // 2nd argument (linea)
        movl $msg, %eax   // 1st argument (msg)
        // (3) Invocar a vga_write2()
        call vga_write_cyan
        // (4) Restaurar registros.
        pop %ecx
        pop %edx
        pop %eax
        iret
.data
msg:
        .asciz "Se divide por ++ebx"
//-----------------------------------------------------------------------------
```

interrupts.c

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

#define IDTSIZE 256
```

```
#define LOW_MASK 0xFFFF
#define DESP 16

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

static struct IDTR idtr;
static struct Gate idt[IDTSIZE];

// Multiboot siempre define "8" como el segmento de código.
// (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;
//------------------------------------------------------------------------------
// IDT INIT
//------------------------------------------------------------------------------
void idt_init(void){

    // (1) Instalar manejadores ("interrupt service routines").
    idt_install(T_BRKPT, breakpoint);
    idt_install(T_DIVIDE, divzero); // Ej: kern2-div

    idtr.base = (uintptr_t) idt;
    idtr.limit = (sizeof(idt) - 1);

    asm("lidt %0" : : "m"(idtr));
}
//------------------------------------------------------------------------------
// IDT INSTALL
//------------------------------------------------------------------------------
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 & LOW_MASK;
    idt[n].off_31_16 = addr >> DESP;

    idt[n].present = 1;
}
//------------------------------------------------------------------------------
```

```
// IRQ REAMP
//-----------------------------------------------------------------------------
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);
}
//-----------------------------------------------------------------------------
// IRQ INIT
//-----------------------------------------------------------------------------
void irq_init() {
    // (1) Redefinir códigos para IRQs.
    irq_remap();

    // (2) Instalar manejadores.
    idt_install(T_TIMER, timer_asm);
    idt_install(T_KEYBOARD, ack_irq);

    // (3) Habilitar interrupciones.
    asm("sti");
}
//-----------------------------------------------------------------------------
```

interrupts.h

```
#ifndef INTERRUPTS_H
#define INTERRUPTS_H

#include <stdint.h>

// IDTR Register (see IA32-3A, Â§6.10 INTERRUPT DESCRIPTOR TABLE).
// IDT puede albergar 256 descriptors como maximo.
// La IDT solo debe albergar los descriptores necesarios para describir las interrupciones
// y excepciones que pueden llegar a ocurrir.
// Cada descriptor (entrada de la tabla) de la IDT tiene un tamaño fijo de 8 bytes.
// Los descriptores vacios de la IDT deben tener el present flag seteado en 0.
// El IDT base addres debe estar alineada a 8 bytes.
```

```
// El valor del limite debe ser siempre (8*N - 1).
// El registro de la IDT (IDTR) contiene la informacion de la IDT
struct IDTR {
    uint16_t limit;  // Limit: tiene tamaño 16 bits
    uint32_t base;   // Base address: tiene tamaño 32 bits
} __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
```

kern2.c

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

#define USTACK_SIZE 4096
static uint8_t stack1[USTACK_SIZE] __attribute__((aligned(4096)));
static uint8_t stack2[USTACK_SIZE] __attribute__((aligned(4096)));
//------------------------------------------------------------------------------
//  TWO STACKS C
//------------------------------------------------------------------------------
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:
    // vga_write("vga_write() from stack1", 15, 0x57);

    // AYUDA 1: se puede usar alguna forma de pre- o post-
    // incremento/decremento, según corresponda:
    //
    //     *(a++) = ...
    //     *(++a) = ...
    //     *(a--) = ...
    //     *(--a) = ...
    *(--a) = 0x57;  // color
    *(--a) = 15;  // linea
    *(--a) = (uintptr_t) "vga_write() from stack1";  // mensaje

    // AYUDA 2: para apuntar a la cadena con el mensaje,
    // es suficiente con el siguiente cast:
```

```
    //
    //   ... a ... = (uintptr_t) "vga_write() from stack1";

    // Preparar, en s2, la llamada:
    // vga_write("vga_write() from stack2", 16, 0xD0);

    // AYUDA 3: para esta segunda llamada, usar esta forma de
    // asignación alternativa:
    b -= 3;
    b[0] = (uintptr_t) "vga_write() from stack2";
    b[1] = 16;
    b[2] = 0xD0;

    // Primera llamada usando task_exec().
    task_exec((uintptr_t) vga_write, (uintptr_t) a);

    // Segunda llamada con ASM directo. Importante: no
    // olvidar restaurar el valor de %esp al terminar, y
    // compilar con: -fasm -fno-omit-frame-pointer.
    //asm("...; call *%1; ..."
    asm("movl %0, %%esp; call *%1; movl %%ebp, %%esp"
    : /* no outputs */
    : "r"(b), "r"(vga_write));
}
//-----------------------------------------------------------------------------
//  KMAIN
//-----------------------------------------------------------------------------
void kmain(const multiboot_info_t *mbi) {
    if (mbi == NULL) vga_write("mbi == NULL", 10, 0x70);
    vga_write("kern2 loading.............", 8, 0x70);
    two_stacks();
    two_stacks_c();

    contador_run();  // Nueva llamada ej. kern2-swap.

    /*
     * Por último, como "bootstrap" del planificador, se necesita una llamada a
     * la función sched_init() desde kmain(), antes de las llamadas a
     * idt_init()/irq_init(). Esto se necesita para que haya una tarea inicial
     * en ejecución.
    */
    sched_init();  // Desalojo: Ej: kern2-task
    // Código ejercicio kern2-idt.
    idt_init();   // (a)
    irq_init();   // Ej: kern2-irq
    asm("int3");  // (b)
```

```
    /* Ej: kern2-div */
    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);
    /* Ej: kern2-div */

    contador_spawn();

    if (mbi->flags & MULTIBOOT_INFO_CMDLINE) {
        char buf[256] = "cmdline: ";
        char *cmdline = (void *) mbi->cmdline;
        strlcat(buf, cmdline, strlen(buf) + strlen(cmdline) + 1);
        vga_write(buf, 9, 0x07);
    }

    char mem[256] = "Physical memory: ";
    char tmp[64] = "";
    uint32_t memSize = mbi->mem_upper - mbi->mem_lower;
    memSize = (memSize >> 10) + 1;

    if (fmt_int(memSize, tmp, sizeof tmp)) {
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, "MiB total (", sizeof mem);
    }
    if (fmt_int(mbi->mem_lower, tmp, sizeof tmp)) {
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, " base, ", sizeof mem);
    }
    if (fmt_int(mbi->mem_upper, tmp, sizeof tmp)) {
        strlcat(mem, tmp, sizeof mem);
        strlcat(mem, "KiB extended)", sizeof mem);
    }
    vga_write(mem, 10, 0x07);

    while (1) asm("hlt");
}
//------------------------------------------------------------------------------
```

multiboot.h

```
/* 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 "lib/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;
    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 {
    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.c

```
#include "lib/stdbool.h"
#include "lib/stddef.h"
#include "decls.h"
#include "sched.h"

#define MAX_TASK 10
// IF Bandera de interrupcion habilitada (bit 9: 0010 0000 0000)
#define IF 0x200

static struct Task Tasks[MAX_TASK];
static struct Task *current;
//-----------------------------------------------------------------------------
// SCHED INIT
//-----------------------------------------------------------------------------
void sched_init() {
    current = &Tasks[0];
    current->status = RUNNING;
}
//-----------------------------------------------------------------------------
// SPAWN
//-----------------------------------------------------------------------------
void spawn(void (*entry)(void)) {
    /*
    Una función spawn() que, dada una función o entry point, deja preparado un
    struct Task para que la tarea entre en ejecución.
    La función spawn() debe:

        *Encontrar en el arreglo global Tasks, una entrada con estado FREE

        *Cambiar su status a READY

        *Inicializar todos sus registros a cero, excepto %cs, %eip y %eflags.
        En particular %eflags debe contener el flag IF, o las interrupciones de
        reloj no se habilitarán al entrar la tarea en ejecución.
    */
    /*
    Te respondo por partes:

        El entry point se coloca, de manera similar a ejercicios anteriores,
        para que ret/iret funcione: "en el tope de la pila una vez hecho pop de
        los registros de propósito general."

        En este caso es más fácil porque tenés el miembro "eip" del struct
```

```
        diciéndote dónde hay que almacenarlo.

        Fijate que el "taskframe" es siempre un puntero, y siempre apuntará a
        alguna región de la pila.

        Dicho de otra manera: en ejercicios anteriores, para todas las tareas
        tuvimos que reservar un stack; en este ejercicio, se reserva
        directamente como parte del struct (podría haber sido en cualquier otro
        sitio, pero esto es más fácil).

        Cuando sched() recibe un puntero a TaskFrame, podés imaginar también que
        recibiera un void* o un uintptr_t*: lo que recibe es el puntero al tope
        de la pila. Ocurre que, en esta configuración, el tope de la pila de una
        tarea que no está en ejecucións siempre "coincide" con un TaskFrame, de
        ahí su uso.

        Técnicamente sched() no necesita que sea un TaskFrame*.

        Espero que esto ayude a aclarar tus dudas, para nuevas consultas no
        dudes en escribirnos de nuevo.
    */
    int i = 0;
    while (Tasks[i].status != FREE) i++;
    if (i == MAX_TASK) return;
    Tasks[i].status = READY;
    uint8_t* stack = &Tasks[i].stack[USTACK_SIZE];
    // cada uint32_t son cuatro casilleros de uint8_t y ademas son 11 porque
    //  uint16_t cs y uint16_t padding cuentan como uno de uint32_t
    size_t size = sizeof(struct TaskFrame);  // size deberia ser 44 = 11*4
    stack -= size;
    Tasks[i].frame = (struct TaskFrame *) (stack);
    Tasks[i].frame->edi = 0;
    Tasks[i].frame->esi = 0;
    Tasks[i].frame->ebp = 0;
    Tasks[i].frame->esp = 0;
    Tasks[i].frame->ebx = 0;
    Tasks[i].frame->edx = 0;
    Tasks[i].frame->ecx = 0;
    Tasks[i].frame->eax = 0;
    Tasks[i].frame->padding = 0;
    Tasks[i].frame->eflags = Tasks[i].frame->eflags | IF;
    Tasks[i].frame->eip = (uint32_t) entry;
}
//------------------------------------------------------------------------------
// SCHED
//------------------------------------------------------------------------------
```

```
void sched(struct TaskFrame *tf) {
    struct Task *new = 0;
    struct Task *old = current;
    /*
    Encontrar, de manera round-robin, la siguiente tarea que se encuentra en
    estado READY. Una posible manera es encontrar en el arreglo la tarea en
    ejecución, y buscar a partir de ahí:
    */

    static int cuenta = 0;
    print((uint64_t) ++cuenta, 6);

    bool foundCurrent = false;
    bool foundReady = false;
    for (int i = 0; i < MAX_TASK; i++) {
        if (Tasks[i].status == RUNNING && !foundCurrent) {
            foundCurrent = true;
            old = &Tasks[i];
        } else if (Tasks[i].status == READY && foundCurrent) {
            foundReady = true;
            new = &Tasks[i];
            break;
        }
    }
    /*
    Si se la encuentra, se debe poner old->status a READY y guardar en
    old->frame el frame recibido como parámetro; actualizar la variable global
    current y en current->status poner RUNNING. Por último, para ejecutar el
    cambio, se puede usar el siguiente assembler:
    */
    if (!foundReady) return;
    old->status = READY;
    old->frame = tf;
    current = new;
    current->status = RUNNING;
    asm("movl %0, %%esp\n"
        "popa\n"
        "iret\n"
    :
    : "g"(current->frame)
    : "memory");
}
//------------------------------------------------------------------------------
```

sched.h

```
#ifndef SCHED_H
#define SCHED_H

#include "lib/stdint.h"

#define USTACK_SIZE 4096

enum TaskStatus {
    FREE = 0,
    READY = 1,
    RUNNING = 2,
    DYING = 3,
};

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);

#endif
```

stacks.S

```
#define USTACK_SIZE 4096
#define STACK_DIR -12
#define FIRST_ARG -12
#define SECOND_ARG -8
#define THIRD_ARG -4
//-------------------------------------------------------------------------------
// TWO STACKS
//-------------------------------------------------------------------------------
.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"

// esqueleto de implementación:
.text
.globl two_stacks
two_stacks:
        // Preámbulo estándar
        push %ebp
        movl %esp, %ebp

        // Registros para apuntar a stack1 y stack2.
        mov $stack1_top, %eax
        mov $stack2_top, %edi   // Decidir qué registro usar.
        // (Cuando use %ecx no andaba!!! preguntar por que)
        // no usar %ebx porque ahi esta el argumento de la
        // funcion kmain, antes lo use y commandline no funcionaba

        // 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, THIRD_ARG(%eax)
```

```
        movl $0x90, THIRD_ARG(%edi)

        movl $12, SECOND_ARG(%eax)
        movl $13, SECOND_ARG(%edi)

        movl $msg1, FIRST_ARG(%eax)
        movl $msg2, FIRST_ARG(%edi)

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

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

        // Realizar segunda llamada con stack2.
        leal STACK_DIR(%edi), %esp
        call vga_write

        // Restaurar registros callee-saved, si se usaron.

        leave
        ret
//---------------------------------------------------------------------------
```

tasks.S

```
#define FIRST_ARG 8
#define SECOND_ARG 12
//---------------------------------------------------------------------------
// TASK EXEC
//---------------------------------------------------------------------------
.text
.globl task_exec
task_exec:
        push %ebp
        movl %esp, %ebp
        movl SECOND_ARG(%ebp), %esp
        call *FIRST_ARG(%ebp)
        movl %ebp, %esp
        leave
```

```
        ret
//-------------------------------------------------------------------------------
// TASK SWAP
//-------------------------------------------------------------------------------
.text
.globl task_swap
task_swap:
        // void task_swap(uintptr_t *esp);
        // Pone en ejecución la tarea cuyo stack está en ‘*esp’, cuyo
        // valor se intercambia por el valor actual de %esp. Guarda y
        // restaura todos los callee-called registers.
        //  Registros que son callee-saved:
        // EBX - EBP - EDI - ESI
        // 24(%esp): Valor del stack antes de llamar a la funcion
        // 20(%esp): Argumento de la funcion uintptr_t *esp
        // 16(%esp): direccion de retorno
        push %edi               // 12(%esp)
        push %ebp               // 8(%esp)
        push %esi               // 4(%esp)
        push %ebx               // 0(%esp)

        mov  20(%esp), %edx
        mov  %esp, %eax
        mov  (%edx), %esp
        mov  %eax, (%edx)

        pop %ebx
        pop %esi
        pop %ebp
        pop %edi
        ret
//-------------------------------------------------------------------------------
```

write.c

```
#include "decls.h"
#define VGABUF ((volatile char *) 0xb8000)
#define MAX_X 80
#define MAX_Y 25
//-------------------------------------------------------------------------------
// VGA WRITE
//-------------------------------------------------------------------------------
// Se escribe la cadena en la línea indicada de la pantalla
```

```
// (si linea es menor que cero, se empieza a contar desde abajo).
void vga_write(const char *s, int8_t linea, uint8_t color) {
    int vgaPsoition;
    volatile char *buf = VGABUF;
    if (linea < 0) {
        vgaPsoition = MAX_Y + linea;
    } else {
        vgaPsoition = linea;
    }
    int pos = 0;
    buf += 2*vgaPsoition*MAX_X;
    while (s[pos] != '\0') {
        *buf++ = s[pos];
        *buf++ = color;
        pos++;
    }
}
//------------------------------------------------------------------------------
// INT WIDTH
//------------------------------------------------------------------------------
static size_t int_width(uint64_t val) {
    uint64_t n = val;
    size_t digits = 0;
    while(n > 0) {
        digits++;
        n = n / 10;
    }
    return digits;
}
//------------------------------------------------------------------------------
// FMT INT
//------------------------------------------------------------------------------
// Escribe en ‘s’ el valor de ‘val’ en base 10 si su anchura
// es menor que ‘bufsize’. 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) {
    if (val == 0) {
        s[0] = (char) 48;
        s[1] = '\0';
        return true;
    }
    size_t l = int_width(val);
    if (l >= bufsize)  // Pregunta: ¿por qué no "l > bufsize"?
        return false;
    uint64_t n = val;
    size_t pos = l;
```

```
    s[pos] = '\0';
    while(n > 0) {
        pos--;
        s[pos] = (char) ((n % 10) + 48);
        n = n / 10;
    }
    return true;
}
//-----------------------------------------------------------------------------
// VGA WRITE CYAN
//-----------------------------------------------------------------------------
void __attribute__((regparm(2))) vga_write_cyan(const char *s, int8_t linea) {
    vga_write(s, linea, 0xB0);
}
//-----------------------------------------------------------------------------
// PRINT
//-----------------------------------------------------------------------------
void print(uint64_t value, int8_t line) {
    char buf[256];
    fmt_int(value, buf, 256);
    vga_write(buf, line, 0x2F);
    buf[0] = '\0';
}
//-----------------------------------------------------------------------------
```