Universidad de Buenos Aires

Sistemas operativos (75.08)

Kern 2

Darius Maitia - 95436

Luis Tejerina - 96629

Contents

1 Creación de stacks en el kernel

1.1 kern2-stack

Explicar: ¿qué significa "estar alineado"?

Significa que la dirección incial de memoria alocada (en este caso para el stack) es múltiplo del número indicado. Por ejemplo, si se está alineado a 4KB, se está diciendo que la dirección puede ser: 0, 4KB, 8KB, 12KB...

Mostrar la sintaxis de C/GCC para alinear a 32 bits el arreglo kstack anterior.

```
unsigned unsigned char kstack[8192] __attribute__ ((aligned (32)));
```

¿A qué valor se está inicializando kstack? ¿Varía entre la versión C y la versión ASM? (Leer la documentación de as sobre la directiva .space.)

Kstack se inicializa en cero. Efectivamente varía entre la versión C y la versión ASM; en C no se setea en cero el stack mientras que en ASM si lo hace.

Explicar la diferencia entre las directivas .align y .p2align de as, y mostrar cómo alinear el stack del kernel a 4 KiB usando cada una de ellas.

Align alinea al número indicado y .p2align alinea a 2 elevado a la potencia indicada. De este modo para alinear el stack del kernel a 4kb deberíamos escribir .align 4096 o .p2align 12.

1.2 kern2-cmdline

Mostrar cómo implementar la misma concatenación, de manera correcta, usando strncat(3)

```
void kmain(const multiboot_info_t *mbi){
    vga_write("NestorKernel loading....", 2, WHITE_BLUE);

     if (mbi->flags & MULTIBOOT_INFO_CMDLINE) {
        char buf[CMD_BUF_SIZE] = "cmdline: ";
        char *cmdline = (void *) mbi->cmdline;

        strncat(buf, cmdline, CMD_BUF_SIZE - strlen(buf));
        vga_write(buf, 9, WHITE_BLUE);
    }
}
```

Explicar cómo se comporta strlcat(3) si, erróneamente, se declarase buf con tamaño 12. ¿Introduce algún error el código?

Strlcat escribe en el buffer en lo que queda del buffer de destino libre para escribir. Si lo que hay que escribir (contactenar) es más que el espacio libre del buffer, corta en el límite sin cometer un segmentation fault. En todos los casos siempre se ocupa de finalizar la cadena con un barra cero.

Compilar el siguiente programa, y explicar por qué se imprimen dos líneas distintas, en lugar de la misma dos veces:

```
#include <stdio.h>

static void printf_sizeof_buf(char buf[256]) {
    printf("sizeof buf = %zu\n", sizeof buf);
}

int main(void) {
    char buf[256];
    printf("sizeof buf = %zu\n", sizeof buf);
    printf_sizeof_buf(buf);
}
```

Al ejecutar este código vemos por la consola:
sizeof buf = 256
sizeof buf = 8

En C no se pueden pasar bloques de memoria como parámetro, se pasan punteros a bloques de memoria. Al ser el parámetro de la funcion printf_sizeof_buf un char buf[256], en sí ese parámetro es un puntero a un array en memoria de 256 chars. Como tal, sizeof buf adentro de esa función va a ser el tamaño de un puntero, en este caso 8 bytes.

2 Concurrencia cooperativa

2.1 kern2-regcall

Mostrar con objdump -d el código generado por GCC para la siguiente llamada a vga_write2() desde la función principal:

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

    two_stacks();
    two_stacks_c();

    vga_write2("Funciona vga_write2?", 18, 0xE0);
}
```

El código generado por GCC es:

```
lea     0x0,%eax
mov     $0x12,%edx
mov     $0xe0,%ecx
call    56 <kmain+0x56>
```

Podemos observar que utiliza el pasaje de argumentos por registros.

2.2 kern2-swap

Explicar, para el stack de cada contador, cuántas posiciones se asignan, y qué representa cada una.
Para el stack del primer contador se asignan 3 posiciones, una para cada argumento de la llamada a contador_yield en orden inverso:

```
//contador_yield(100, 0, 0x2F);
*(--stack1_ptr) = 0x2F;
*(--stack1_ptr) = 0;
*(--stack1_ptr) = 100;
```

Para el stack del segundo contador se asignan, en primer lugar los argumentos de contador_yield ademas de la direccion de esa funcion para que al retornar de task_swap la primera vez retorne a la funcion contador_yield, además de una extra para seguir la convencio de llamadas, luego se encolan los registros callee saved para simular una llamada previa a task_swap.

```
 //contador_yield(100, 1, 0x4F);
*(--stack2_ptr) = 0x4F;
*(--stack2_ptr) = 1;
*(--stack2_ptr) = 100;
*(--stack2_ptr) = &contador_yield;  //extra return address
*(--stack2_ptr) = &contador_yield;
asm ("movl %%edi, %0" : "=r" (*(--stack2_ptr)) );
asm ("movl %%esi, %0" : "=r" (*(--stack2_ptr)) );
asm ("movl %%ebp, %0" : "=r" (*(--stack2_ptr)) );
asm ("movl %%ebx, %0" : "=r" (*(--stack2_ptr)) );
```

3 Interrupciones: reloj y teclado

3.1 kern2-idt

¿Cuántos bytes ocupa una entrada en la IDT?

Una entrada de la IDT ocupa 8 bytes.

¿Cuántas entradas como máximo puede albergar la IDT?

La IDT puede albergar a lo sumo 256 entradas.

¿Cuál es el valor máximo aceptable para el campo limit del registro IDTR?

El registro IDTR se compone de 48 bits, 16 para el campo IDT Limit y el resto para el campo IDT Base Address.

Indicar qué valor exacto tomará el campo limit para una IDT de 64 descriptores solamente.

Para direccionar 64 descriptores, harían falta 6 bits (2**6 = 64) pero como que cada entrada ocupa 4 bytes, necesitamos 2 bits de offset, con lo que en total hacen falta 8 bits, con los dos menos significativos seteados en 0. En definitiva, el campo limit en binario tomará la forma de 11111100.

Consultar la sección 6.1 y explicar la diferencia entre interrupciones (§6.3) y excepciones (§6.4).

Las interrupciones y las excepciones ambas alteran el flujo del programa. Sin embargo las interrupciones se usan para manejar eventos externos (como el input del teclado) y las excepciones son usadas para manejar fallas en las instrucciones (como una división por cero).

Las interrupciones son además manejadas por el procesador después de finalizar la instrucción que se estaba ejecutando, son también eventos asincrónicos generados por hardware (aunque no siempre) que no está sincronizado con las instrucciones del procesador. Por el contrario las excepciones si son eventos sincrónicos generados cuando el procesador detecta una condición predefinida mientras ejecuta instrucciones.

3.2 kern2-isr

3.2.1 Sesión de GDB versión A

Paso 1

- la impresión de la siguiente instrucción ejecutando display/i $pc muestra:
  => 0xfff0: add %al,(%eax)

Paso 2

- Las instrucciones de código assembler mostradas por el comando disas al momento del punto de interrupción:

```
(gdb) disas
Dump of assembler code for function kmain:
   0x001004e0 <+0>: push   %ebp
   0x001004e1 <+1>: mov    %esp,%ebp
   0x001004e3 <+3>: push   %esi
   0x001004e4 <+4>: sub    $0x134,%esp
   0x001004ea <+10>: mov    0x8(%ebp),%eax
   0x001004ed <+13>: lea    0x100c3c,%ecx
   0x001004f3 <+19>: mov    $0x2,%edx
   0x001004f8 <+24>: mov    $0x1f,%esi
   0x001004fd <+29>: mov    %eax,-0x8(%ebp)
   0x00100500 <+32>: mov    %ecx,(%esp)
   0x00100503 <+35>: movl   $0x2,0x4(%esp)
```

```
   0x0010050b <+43>: movl   $0x1f,0x8(%esp)
   0x00100513 <+51>: mov    %esi,-0x110(%ebp)
   0x00100519 <+57>: mov    %edx,-0x114(%ebp)
   0x0010051f <+63>: call   0x1006a0 <vga_write>
---Type <return> to continue, or q <return> to quit---
   0x00100524 <+68>: mov    -0x8(%ebp),%eax
   0x00100527 <+71>: mov    (%eax),%eax
   0x00100529 <+73>: and    $0x4,%eax
   0x0010052e <+78>: cmp    $0x0,%eax
   0x00100533 <+83>: je     0x1005c8 <kmain+232>
   0x00100539 <+89>: mov    $0x100,%eax
   0x0010053e <+94>: lea    -0x108(%ebp),%ecx
   0x00100544 <+100>: lea    0x100c9e,%edx
   0x0010054a <+106>: mov    %ecx,%esi
   0x0010054c <+108>: mov    %esi,(%esp)
   0x0010054f <+111>: mov    %edx,0x4(%esp)
   0x00100553 <+115>: movl   $0x100,0x8(%esp)
   0x0010055b <+123>: mov    %eax,-0x118(%ebp)
   0x00100561 <+129>: mov    %ecx,-0x11c(%ebp)
   0x00100567 <+135>: call   0x1006f0 <memcpy>
   0x0010056c <+140>: mov    -0x8(%ebp),%eax
---Type <return> to continue, or q <return> to quit---
   0x0010056f <+143>: mov    0x10(%eax),%eax
   0x00100572 <+146>: mov    %eax,-0x10c(%ebp)
   0x00100578 <+152>: mov    -0x10c(%ebp),%eax
   0x0010057e <+158>: mov    -0x11c(%ebp),%ecx
   0x00100584 <+164>: mov    %ecx,(%esp)
   0x00100587 <+167>: mov    %eax,0x4(%esp)
   0x0010058b <+171>: movl   $0x100,0x8(%esp)
   0x00100593 <+179>: call   0x100bc0 <strlcat>
   0x00100598 <+184>: mov    %eax,-0x120(%ebp)
   0x0010059e <+190>: call   0x100044 <two_stacks>
   0x001005a3 <+195>: call   0x1005e0 <two_stacks_c>
   0x001005a8 <+200>: call   0x1003c0 <idt_init>
=> 0x001005ad <+205>: lea    0x100c55,%eax
   0x001005b3 <+211>: mov    $0x12,%edx
   0x001005b8 <+216>: mov    $0xe0,%ecx
   0x001005bd <+221>: int3
---Type <return> to continue, or q <return> to quit---
   0x001005be <+222>: call   0x100030 <vga_write2>
   0x001005c3 <+227>: call   0x1000c0 <contador_run>
   0x001005c8 <+232>: add    $0x134,%esp
   0x001005ce <+238>: pop    %esi
   0x001005cf <+239>: pop    %ebp
   0x001005d0 <+240>: ret
```

el valor de %esp (print $esp)

```
(gdb) print $esp
$1 = (void *) 0x103eb8
```

- el valor de (%esp) (x/xw $esp)

```
(gdb) x/xw $esp
0x103eb8: 0x00103ee8
```

- el valor de $cs

```
(gdb) print $cs
$2 = 8
```

- el resultado de print $eflags y print/x $eflags

```
(gdb) print $eflags
$3 = [ AF ]
(gdb) print/x $eflags
$4 = 0x12
```

Paso 3

Ejecutar la instrucción int3 mediante el comando de GDB stepi. La ejecución debería saltar directamente a la instrucción test %eax, %eax; en ese momento:

- imprimir el valor de %esp; ¿cuántas posiciones avanzó?

```
(gdb) print $esp
$7 = (void *) 0x103eac
```

Avanzó 12 posiciones.
si avanzó N posiciones, mostrar (con x/Nwx $sp) los N valores correspondientes

```
(gdb) x/12wx $esp
0x103eac: 0x001005be 0x00000008 0x00000012 0x00103ee8
0x103ebc: 0x0010c000 0x00000100 0x00000000 0x00000000
0x103ecc: 0x00000000 0x00000012 0x00103ee8 0x00000100
```

- mostrar el valor de $eflags

```
(gdb) print $eflags
$8 = [ AF ]
```

¿Qué representa cada valor?

  - 0x00103eac: Error Code
  - 0x001005be: EIP
  - 0x00000008: CS
  - 0x00000012: EFLAGS

Paso 4

Avanzar una instrucción más con stepi, ejecutando la instrucción TEST. Mostrar, como anteriormente, el valor del registro EFLAGS (en dos formatos distintos, usando print y print/x).

```
(gdb) print $eflags
$2 = [ PF ]
(gdb) print /x $eflags
$10 = 0x6
```

Paso 5

Avanzar, por última vez, una instrucción, de manera que se ejecute IRET para la sesión A, y RET para la sesión B. Mostrar, de nuevo lo pedido que en el punto 1; y explicar cualquier diferencia entre ambas versiones A y B.

```
1: x/i $pc
=> 0x1005be <kmain+222>: call   0x100030 <vga_write2>
```

3.2.2 Sesión de GDB versión B

Paso 1

- la impresión de la siguiente instrucción ejecutando display/i $pc muestra:
  => 0xfff0: add %al,(%eax)

Paso 2

- Las instrucciones de código assembler mostradas por el comando disas al momento del punto de interrupción:

```
(gdb) disas
Dump of assembler code for function kmain:
   0x001004e0 <+0>: push   %ebp
   0x001004e1 <+1>: mov    %esp,%ebp
   0x001004e3 <+3>: push   %esi
   0x001004e4 <+4>: sub    $0x134,%esp
   0x001004ea <+10>: mov    0x8(%ebp),%eax
   0x001004ed <+13>: lea    0x100c3c,%ecx
   0x001004f3 <+19>: mov    $0x2,%edx
   0x001004f8 <+24>: mov    $0x1f,%esi
   0x001004fd <+29>: mov    %eax,-0x8(%ebp)
   0x00100500 <+32>: mov    %ecx,(%esp)
   0x00100503 <+35>: movl   $0x2,0x4(%esp)
   0x0010050b <+43>: movl   $0x1f,0x8(%esp)
   0x00100513 <+51>: mov    %esi,-0x110(%ebp)
   0x00100519 <+57>: mov    %edx,-0x114(%ebp)
   0x0010051f <+63>: call   0x1006a0 <vga_write>
   0x00100524 <+68>: mov    -0x8(%ebp),%eax
   0x00100527 <+71>: mov    (%eax),%eax
   0x00100529 <+73>: and    $0x4,%eax
   0x0010052e <+78>: cmp    $0x0,%eax
   0x00100533 <+83>: je     0x1005c8 <kmain+232>
   0x00100539 <+89>: mov    $0x100,%eax
   0x0010053e <+94>: lea    -0x108(%ebp),%ecx
   0x00100544 <+100>: lea    0x100c9e,%edx
   0x0010054a <+106>: mov    %ecx,%esi
   0x0010054c <+108>: mov    %esi,(%esp)
   0x0010054f <+111>: mov    %edx,0x4(%esp)
   0x00100553 <+115>: movl   $0x100,0x8(%esp)
   0x0010055b <+123>: mov    %eax,-0x118(%ebp)
   0x00100561 <+129>: mov    %ecx,-0x11c(%ebp)
   0x00100567 <+135>: call   0x1006f0 <memcpy>
   0x0010056c <+140>: mov    -0x8(%ebp),%eax
   0x0010056f <+143>: mov    0x10(%eax),%eax
   0x00100572 <+146>: mov    %eax,-0x10c(%ebp)
   0x00100578 <+152>: mov    -0x10c(%ebp),%eax
   0x0010057e <+158>: mov    -0x11c(%ebp),%ecx
   0x00100584 <+164>: mov    %ecx,(%esp)
   0x00100587 <+167>: mov    %eax,0x4(%esp)
   0x0010058b <+171>: movl   $0x100,0x8(%esp)
   0x00100593 <+179>: call   0x100bc0 <strlcat>
---Type <return> to continue, or q <return> to quit---
   0x00100598 <+184>: mov    %eax,-0x120(%ebp)
   0x0010059e <+190>: call   0x100044 <two_stacks>
   0x001005a3 <+195>: call   0x1005e0 <two_stacks_c>
   0x001005a8 <+200>: call   0x1003c0 <idt_init>
=> 0x001005ad <+205>: lea    0x100c55,%eax
   0x001005b3 <+211>: mov    $0x12,%edx
   0x001005b8 <+216>: mov    $0xe0,%ecx
   0x001005bd <+221>: int3
   0x001005be <+222>: call   0x100030 <vga_write2>
   0x001005c3 <+227>: call   0x1000c0 <contador_run>
   0x001005c8 <+232>: add    $0x134,%esp
   0x001005ce <+238>: pop    %esi
   0x001005cf <+239>: pop    %ebp
   0x001005d0 <+240>: ret
End of assembler dump.
```

el valor de %esp (print $esp)

```
(gdb) print $esp
$1 = (void *) 0x103eb8
```

- el valor de (%esp) (x/xw $esp)

```
(gdb) x/xw $esp
0x103eb8: 0x00103ee8
```

- el valor de $cs

```
(gdb) print $cs
$2 = 8
```

- el resultado de print $eflags y print/x $eflags

```
(gdb) print $eflags
$3 = [ AF ]
(gdb) print/x $eflags
$4 = 0x12
```

Paso 3

Ejecutar la instrucción int3 mediante el comando de GDB stepi. La ejecución debería saltar directamente a la instrucción test %eax, %eax; en ese momento:

- imprimir el valor de %esp; ¿cuántas posiciones avanzó?

```
(gdb) print $esp
$7 = (void *) 0x103eac
```

Avanzó 12 posiciones.
si avanzó N posiciones, mostrar (con x/Nwx $sp) los N valores correspondientes

```
(gdb) x/12wx $esp
0x103eac: 0x001005be 0x00000008 0x00000012 0x00103ee8
0x103ebc: 0x0010c000 0x00000100 0x00000000 0x00000000
0x103ecc: 0x00000000 0x00000012 0x00103ee8 0x00000100
```

- mostrar el valor de $eflags

```
(gdb) print $eflags
$8 = [ AF ]
```

- ¿Qué representa cada valor?
  - 0x00103eac: Error Code
  - 0x001005be: EIP
  - 0x00000008: CS
  - 0x00000012: EFLAGS

Paso 4

Avanzar una instrucción más con stepi, ejecutando la instrucción TEST. Mostrar, como anteriormente, el valor del registro EFLAGS (en dos formatos distintos, usando print y print/x).

```
(gdb) print $eflags
$2 = [ PF ]
(gdb) print /x $eflags
$10 = 0x6
```

Paso 5

Avanzar, por última vez, una instrucción, de manera que se ejecute IRET para la sesión A, y RET para la sesión B. Mostrar, de nuevo lo pedido que en el punto 1; y explicar cualquier diferencia entre ambas versiones A y B.

```
1: x/i $pc
=> 0x1005be <kmain+222>: call   0x100030 <vga_write2>
```

Observamos que los resultados para la versión A y la versión B son idénticos.

3.2.3 Versión final de breakpoint()

Para cada una de las siguientes maneras de guardar/restaurar registros en breakpoint, indicar si es correcto (en el sentido de hacer su ejecución "invisible"), y justificar por qué:

- Opción A:
  Este método es seguro ya que guarda todos los registros y los restaura al finalizar la llamada, por lo que hace su ejecución invisible para la función llamadora.
- Opción B:
  Este método también es seguro ya que guarda los registros que son caller safe y los restaura al finalizar lo que tenía que hacer.
- Opción C:
  Este método no es seguro porque no te garantiza que los registros eax, edx y ecx no sean alterados por el código representado por puntos suspensivos.

Responder de nuevo la pregunta anterior, sustituyendo en el código vga_write2 por vga_write.

Sucede lo mismo que para la versión con vga_write2. Para la opción C, nada te garantiza que los registros que no son callee safe (eax, edx, ecx) no sean alterados por vga_write.

3.3 kern2-irq

...

3.4 kern2-div

Explicar el funcionamiento exacto de la línea asm(...) del punto anterior:

- **¿Qué cómputo se está realizando?**
  Se está dividiendo por cero.
- **¿De dónde sale el valor de la variable color**
  El valor de la variable color sale del campo "1"(0xE0), donde el valor asignado es 0xE0.
- **¿Por qué se da el valor 0 a %edx?**
  Porque ante una división por cero, al no haber un valor conocido, se indica como que el resultado es 0 por defecto.

4 Anexo

4.0.1 contador.c

```
#include "lib/decls.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)));

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

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();
    }
}

void contador_run() {
    //contador_yield(100, 0, 0x2F);
    // Inicializar al *tope* de cada pila.
    uintptr_t *stack1_ptr = stack1 + USTACK_SIZE * sizeof(uint8_t);
    uintptr_t *stack2_ptr = stack2 + USTACK_SIZE * sizeof(uint8_t);

    // Preparar, en stack1, la llamada:
    //contador_yield(100, 0, 0x2F);
    *(--stack1_ptr) = 0x2F;
    *(--stack1_ptr) = 0;
    *(--stack1_ptr) = 100;

    // Preparar, en stack2, la llamada:
    //contador_yield(100, 1, 0x4F);
    *(--stack2_ptr) = 0x4F;
    *(--stack2_ptr) = 1;
    *(--stack2_ptr) = 100;
    *(--stack2_ptr) = &contador_yield;  //gcc extra return address
    *(--stack2_ptr) = &contador_yield;
    asm ("movl %%edi, %0" : "=r" (*(--stack2_ptr)) );
    asm ("movl %%esi, %0" : "=r" (*(--stack2_ptr)) );
    asm ("movl %%ebp, %0" : "=r" (*(--stack2_ptr)) );
    asm ("movl %%ebx, %0" : "=r" (*(--stack2_ptr)) );

    // Actualizar la variable estatica 'esp' para que apunte
    // al del segundo contador.
    esp = stack2_ptr;
    // Lanzar primer contador
    task_exec((uintptr_t) contador_yield, (uintptr_t) stack1_ptr);
}
```

4.0.2 handlers.c

```
//
// Created by darius on 21/06/18.
//

#include "decls.h"

static unsigned ticks;

void timer() {
  if (++ticks == 15) {
    vga_write("Transcurrieron 15 ticks", 20, 0x07);
  }
}
```

4.0.3 interrupts.c

```
//
// Created by darius on 18/06/18.
//

//#include <elf.h>

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

#define MAX_DESCRIPTORS 256

// Multiboot siempre define "8" como el segmento de codigo.
// (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;

static struct IDTR idtr;

static struct Gate idt[MAX_DESCRIPTORS] __attribute__ ((aligned (8)));

void idt_init(void){
  idt_install(T_BRKPT, breakpoint);
  idt_install(T_DIVIDE, divzero);

  idtr.base = (uintptr_t) &idt;
  idtr.limit = 8*MAX_DESCRIPTORS - 1;

  asm("lidt %0" : : "m"(idtr));
}

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

  idt[n].present = 1;
}

void irq_init() {
  irq_remap();

  idt_install(T_TIMER, timer_asm);
  idt_install(T_KEYBOARD, ack_irq);

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

4.0.4 kern.c

```
#include "lib/decls.h"
#include "lib/interrupts.h"
#include "lib/multiboot.h"
#include "lib/string.h"
#define CMD_BUF_SIZE 256
#define USTACK_SIZE 4096

void kmain(const multiboot_info_t *mbi) {
  int8_t linea;
  uint8_t color;

  vga_write("NestorKernel loading....", 2, WHITE_BLUE);
```

```
  if (mbi->flags & MULTIBOOT_INFO_CMDLINE) {
    char buf[CMD_BUF_SIZE] = "cmdline: ";
    char *cmdline = (void *) mbi->cmdline;
    // Aqui usar strlcat() para concatenar cmdline a buf.
    strlcat(buf, cmdline, CMD_BUF_SIZE);

    /*strncat(buf, cmdline, CMD_BUF_SIZE - strlen(buf));
    vga_write(buf, 9, WHITE_BLUE);
        vga_write("vga_write() from stack1", 12, 0x17);
      vga_write("vga_write() from stack2", 13, 0x90);*/

    two_stacks();
    two_stacks_c();

    idt_init();
    irq_init();

    asm("int3");

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

    vga_write2("Funciona vga_write2?", 18, 0xE0);
    contador_run();
  }
}

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 *stack1_ptr = stack1 + USTACK_SIZE * sizeof(uint8_t);
  uintptr_t *stack2_ptr = stack2 + USTACK_SIZE * sizeof(uint8_t);

  // Preparar, en stack1, la llamada:
  //vga_write("vga_write() from stack1", 15, 0x57);
  *(--stack1_ptr) = 0x57;
  *(--stack1_ptr) = 15;
  *(--stack1_ptr) = (uintptr_t) "vga_write() from stack1 C";

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

  // AYUDA 3: para esta segunda llamada, usar esta forma de
  // asignacion alternativa:
  stack2_ptr -= 3;
  stack2_ptr[0] = (uintptr_t) "vga_write() from stack2 C";
  stack2_ptr[1] = 16;
  stack2_ptr[2] = 0xD0;

  // Primera llamada usando task_exec().
  task_exec((uintptr_t) vga_write, (uintptr_t) stack1_ptr);

  // Segunda llamada con ASM directo. Importante: no
  // olvidar restaurar el valor de %esp al terminar, y
  // compilar con: -fasm -fno-omit-frame-pointer.
  asm(
  "movl %%esp, %%ebp;"
      "leal 0(%0), %%esp;"
      "call *%1;"
      "movl %%ebp, %%esp;"
  ::"r"(stack2_ptr), "r"(vga_write));
}
```

4.0.5 write.c

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

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

void vga_write(const char *string, int8_t linea, uint8_t color){
    volatile char *vga = VGABUF + SCREEN_WIDTH * linea;
```

```
    while( *string != 0 ) {
        *vga++ = *string++;
        *vga++ = color;
    }
}

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

4.0.6 boot.S

```
#include "lib/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
    // El sp apunta al final del stack
    movl $(kstack + KSTACK_SIZE), %esp
    push %ebp

    // Paso 2: pasar la informacion multiboot a kmain. Si el
    // kernel no arranco via Multiboot, se debe pasar NULL.
    //
    // Usar una instruccion de comparacion (TEST o CMP) para
    // comparar con MULTIBOOT_BOOTLOADER_MAGIC, pero no usar
    // un salto a continuacion, sino una instruccion CMOVcc
    // (copia condicional).
     //   mov src, dst

    // Si en eax esta MULTIBOOT_BOOTLOADER_MAGIC significa que
    //   arrancamos con multiboot */
    test %eax, MULTIBOOT_BOOTLOADER_MAGIC
    // Pongo cero en edx despues pongo ebx (direccion de memoria del struct pedido)
    //   si el cmp anterior dio equals y pusheo edx (0 si no equals o ebx si equals
        )

    movl $0, %edx
    cmove %ebx, %edx
    push %edx

    call kmain
halt:
    hlt
    jmp halt

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

4.0.7 funcs.S

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

        push %ecx
        push %edx
        push %eax
```

```
        call vga_write

        leave
        ret
```

4.0.8 idtentry.S

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

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

        movl $18, %edx
        movl $div0_msg, %eax

        call vga_write_cyan

        leave
        addl $1, %ebx

        iret

.globl ack_irq
ack_irq:
        // Indicar que se manejo la interrupcion.
        movl $ACK_IRQ, %eax
        outb %al, $PIC1
        iret

.globl timer_asm
timer_asm:
        // Guardar registros.
        push %ebp
        movl %esp, %ebp

        call timer
        // Restaurar registros.
        leave
        jmp ack_irq

.globl breakpoint
breakpoint:
        // (1) Guardar registros.
        push %ebp
        movl %esp, %ebp

        // (2) Preparar argumentos de la llamada.


        movl $0xB0, %ecx
        movl $14, %edx
        movl $breakpoint_msg, %eax

        // (3) Invocar a vga_write2()
        call vga_write2
        // (4) Restaurar registros.

        // (5) Finalizar ejecucion del manejador.
        leave
        iret

.data
div0_msg:
        .asciz "Se divide por ++ebx"

breakpoint_msg:
        .asciz "Hello, breakpoint"
```

4.0.9 stack.S

```
// 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"

// stacks.S continuado
.text
.globl two_stacks
two_stacks:
        // Preambulo estandar
        push %ebp
        movl %esp, %ebp

        push %ebx

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

        // 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 ultimos MOV para calcular
        // la direccion deseada de ESP.
        leal -12(%eax), %esp
        call vga_write

        // Restaurar stack original. Es %ebp suficiente?
        movl 0(%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
```

4.0.10 tasks.S

```
.text
.globl task_exec
task_exec:
        // Preambulo estandar
        push %ebp /// <- ebp lo guarda la callee o la caller?
        movl %esp, %ebp
        // Porque es 8 y no 4? Vi en el gdb que en el stack hay algo antes que ni
            idea quien lo metio ahi
        movl 8(%esp), %eax
```

```
        movl 12(%esp), %ebx

        // Realizar primera llamada con stack1. Ayuda: usar LEA
        // con el mismo offset que los ultimos MOV para calcular
        // la direccion deseada de ESP.
        leal 0(%ebx), %esp
        call *%eax

        // Restaurar stack original. Es %ebp suficiente?
        movl 0(%ebp), %esp


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

        leave
        ret

.text
.globl task_swap
task_swap:
        // push calle saved
        push %ebx
        push %ebp
        push %esi
        push %edi

        //intercambiar esp y *esp_parametro
        movl 20(%esp), %eax // eax= esp_parametro
        movl 0(%eax), %edx // edx = *esp_parametro
        movl %esp, 0(%eax) // esp_parametro = esp
        movl %edx, %esp //

        //pop calle saved
        pop %edi
        pop %esi
        pop %ebp
        pop %ebx

        ret
```

4.0.11 lib/decls.h

```
#ifndef KERN2_DECL_H
#define KERN2_DECL_H

#include <stdint.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);

// 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
#define SCREEN_WIDTH 160
#define DEFAULT_COLOR 0x07
#define WHITE_BLUE 0x1F
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
```

4.0.12 lib/interrupts.h

```
#ifndef INTERRUPTS_H
#define INTERRUPTS_H

#include <stdint.h>

// IDTR Register (see IA32-3A, 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, 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, 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
};

#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);
}

#endif
```

4.0.13 lib/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 <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 */
```

4.0.14 lib/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;
}

/** IMPLEMENTADO **/
char* strncat(char *dest, const char *src, size_t n)
{
  size_t dest_len = strlen(dest);
  size_t i;

  for (i = 0 ; i < n && src[i] != '\0' ; i++)
    dest[dest_len + i] = src[i];
  dest[dest_len + i] = '\0';

  return dest;
}


/* 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;
}
```

4.0.15 lib/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 PINTOS_LIB_STRING_H
#define PINTOS_LIB_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);

/** IMPLEMENTADOS **/
char* strncat(char *dest, const char *src, size_t n);
```

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